Computational Hydrodynamics for Astrophysics

Michael Zingale

part of the Open Astrophysics Bookshelf

June 29, 2015

document git version: 6e0249aeeefc . . .

the source for these notes are available online (via `git`): `https://github.com/Open-Astrophysics-Bookshelf/numerical_exercises`

Chapter Listing

Table of Contents

List of Figures

List of Exercises

preface

This text started as notes for new students at Stony Brook University working on projects in computational astrophysics. They are intended to help new students come up to speed on the common methods used in computational hydrodynamics for astrophysical flows. They are written at a level appropriate for upper-level undergraduates. The focus is on discussing the *practical issues* that arise when implementing these methods, with less emphasis on the underlying theory—references are provided to fill in details where necessary.

These are very much a work in progress, likely not spell-checked, and definitely not complete. The page size is formatted for easy reading on a tablet or for 2-up printing in a landscape orientation on letter-sized paper.

This text is part of the Open Astrophysics Bookshelf. Contributions to these notes are welcomed. The LaTeX source for these notes is available online on github at:

`https://github.com/Open-Astrophysics-Bookshelf/numerical_exercises`

Simply fork the notes, hack away, and submit a pull-request to add your contributions. All contributions will be acknowledged in the text.

A PDF version of the notes is always available at:

`http://bender.astro.sunysb.edu/hydro_by_example/CompHydroTutorial.pdf`

These notes are updated at irregular intervals, usually when I have a new student working with me, or if I am using them for a course.

The best way to understand the methods described here is to run them for yourself. There are two sets of example codes that go along with these notes:

1. hydro_examples is a set of simple 1-d, standalone python scripts that illustrate some of the basic solvers. Many of the figures in these notes were created using these codes—the relevant script will be noted in the figure caption.

   You can get this set of scripts from github at:
   `https://github.com/zingale/hydro_examples/`

   References to the scripts in hydro_examples are shown throughout the text as:

hydro_examples: `scriptname`

Clicking on the name of the script will bring up the source code to the script (on github) in your web browser.

More details on the codes available in hydro_examples are described in Appendix A.

2. The pyro code [55] is a 2-d simulation code with solvers for advection, diffusion, compressible and incompressible hydrodynamics, as well as multigrid. A gray flux-limited diffusion radiation hydrodynamics solver is in development. pyro is designed with clarity in mind and to make experimentation easy.

   You can download pyro at:
   `https://github.com/zingale/pyro2/`

   A brief overview of pyro is given in Appendix B, and more information can be found at:
   `http://zingale.github.io/pyro2/`

These notes benefited immensely from numerous conversations and an ongoing collaboration with Ann Almgren, John Bell, & Andy Nonaka—pretty much everything I know about projection methods comes from working with them. Discussions with Alan Calder and Chris Malone have also been influential in the presentation of these notes.

If you find errors, please e-mail me at Michael.Zingale@stonybrook.edu, or issue a pull request to the git repo noted above.

Michael Zingale
Stony Brook University

Authorship

Primary Author

Michael Zingale (Stony Brook)

Contributions

Thank you to the following people for pointing out typos or confusing remarks in the text: Chen-Hung and Chris Malone.

See the git log for full details on contributions. All contributions via pull-requests will be acknowledged here.

Chapter 1

Simulation Overview

1.1 What Is Simulation?

Astronomy is an observational science. Unlike in terrestrial physics, we do not have the luxury of being able to build a model system and do physical experimentation on it to understand the core physics. We have to take what nature gives us. Simulation enables us to build a model of a system and allows us to do virtual experiments to understand how this system reacts to a range of conditions and assumptions.

It's tempting to think that one can download a simulation code, set a few parameters, maybe edit some initial conditions, run, and then have a virtual realization of some astrophysical system that you are interested in. Just like that. In practice, it is not this simple. All simulation codes make approximations—these start even before one turns to the computer, simply by making a choice of what equations are to be solved. Typically, we have a system of PDEs, and we need to convert the continuous functional form of our system into a discrete form that can be represented in the finite memory of a computer. This introduces yet more approximation.

Blindly trusting the numbers that come out of the code is a recipe for disaster. You don't stop being a physicist the moment you execute the code—you job as a computational scientist is to make sure that the code is producing reasonable results, by testing it against known problems and your physical intuition.

Simulations should be used to gain insight and provide a physical understanding. Because the systems we solve are so nonlinear, small changes in the code or the programming environment (compilers, optimization, etc.) can produce large differences in the numbers coming out of the code. That's not a reason to panic. As such its best not to obsess about precise numbers, but rather the trends. To really understand the limits of your simulations, you should do parameter and convergence studies.

There is no "über-code". Every algorithm begins with approximations and has limitations. Comparisons between different codes are important and common in our field, and build confidence in the results that we are on the right track.

To really understand your simulations, you need to know what the code your are using is doing under the hood. This means understanding the core methods used in our field. These notes are designed to provide a basic tour of some of the more popular methods, referring to the key papers for full derivations and details. A companion python code, pyro is available to help.

1.2 Numerical Basics

We assume a familiarity with basic numerical methods, which we summarize below. Any book on numerical methods can provide a deeper discussion of these methods. Some good choices are the texts by Garcia [27] and Pang [41].

1.2.1 Sources of Error

With any algorithm, there are two sources of error we are concerned with: *roundoff error* and *truncation error*.

Roundoff arises from the error inherent in representing a floating point number with a finite number of bits in the computer memory. An excellent introduction to the details of how computers represent numbers is provided in [28].

> **Exercise 1.1:** *To see roundoff error in action, write a program to find the value of ϵ for which $1 + \epsilon = 1$. Start with $\epsilon = 1$ and iterate, halving ϵ each iteration unit $1 + \epsilon = 1$. This last value of ϵ for which this was not true is the* machine epsilon. *You will get a different value for single- vs. double-precision floating point arithmetic.*

Some reorganization of algorithms can help minimize roundoff, e.g. avoiding the subtraction of two very large numbers by factoring as:

$$x^3 - y^3 = (x - y)(x^2 + xy + y^2) \ , \tag{1.1}$$

but roundoff error will always be present at some level.

Truncation error is a feature of an algorithm—we typically approximate an operator or function by expanding about some small quantity. When we throw away higher-order terms, we are truncating our expression, and introducing an error in the representation. If the quantity we expand about truly is small, then the error is

small. A simple example is to consider the Taylor series representation of $\sin(x)$:

$$\sin(x) = \sum_{n=1}^{\infty} (-1)^{n-1} \frac{x^{2n-1}}{(2n-1)!} \tag{1.2}$$

For $|x| \ll 1$, we can approximate this as:

$$\sin(x) \approx x - \frac{x^3}{6} \tag{1.3}$$

in this case, our truncation error has the leading term $\propto x^5$, and we say that our approximation is $\mathcal{O}(x^5)$, or 5^{th}-order accurate.

> **Exercise 1.2:** *We will be concerned with the order-of-accuracy of our methods, and a good way to test whether our method is behaving properly is to perform a convergence test. Consider our 5^{th}-order accurate approximation to $\sin(x)$ above. Pick a range of x's (< 1), and compute the error in our approximation as $\epsilon \equiv \sin(x) - [x - x^3/6]$, and show that as you cut x in half, $|\epsilon|$ reduces by 2^5—demonstrating 5^{th}-order accuracy.*

This demonstration of measuring the error as we vary the size of our small parameter is an example of a *convergence test*.

1.2.2 Differentiation and Integration

For both differentiation and integration, there are two cases we might encounter:

1. We have function values, f_0, f_1, ..., at a discrete number of points, x_0, x_1, ..., and we want to compute the derivative at a point or integration over a range of points.
2. We know a function analytically and we want to construct a derivative or integral of this function numerically. In these notes, we'll be concerned only with the first case.

1.2.3 Differentiation

Consider a collection of equally spaced points, labeled with an index i, with the physical spacing between them denoted Δx. We can express the first derivative of a quantity a at i as:

$$\left.\frac{\partial a}{\partial x}\right|_i \approx \frac{a_i - a_{i-1}}{\Delta x} \tag{1.4}$$

or

$$\left.\frac{\partial a}{\partial x}\right|_i \approx \frac{a_{i+1} - a_i}{\Delta x} \tag{1.5}$$

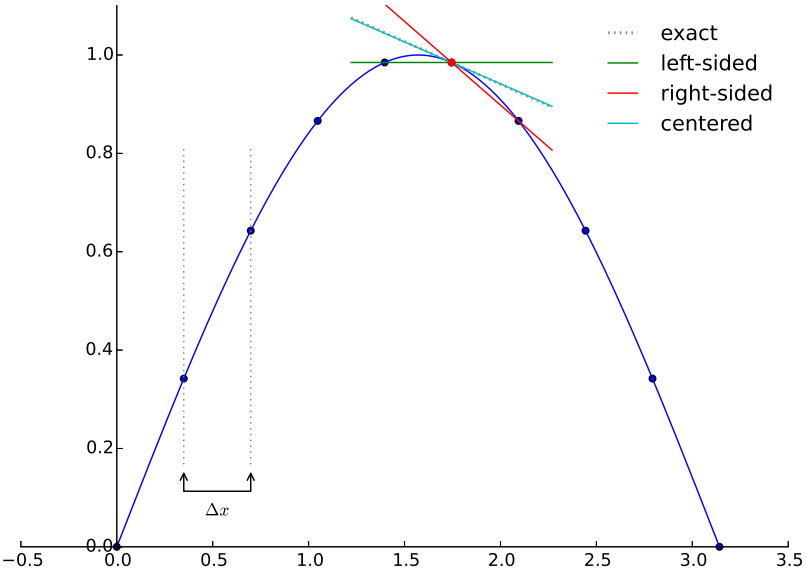

Figure 1.1: A comparison of one-sided and centered difference approximations to the derivative of $\sin(x)$.

(Indeed, as $\Delta x \rightarrow 0$, this is the definition of a derivative from calculus.) Both of these are *one-sided differences*. By Taylor expanding the data about x_i, we see

$$a_{i+1} = a_i + \Delta x \left.\frac{\partial a}{\partial x}\right|_i + \frac{1}{2}\Delta x^2 \left.\frac{\partial^2 a}{\partial x^2}\right|_i + \ldots \tag{1.6}$$

Solving for $\partial a/\partial x|_i$, we see

$$\left.\frac{\partial a}{\partial x}\right|_i = \frac{a_i - a_{i-1}}{\Delta x} + \mathcal{O}(\Delta x) \tag{1.7}$$

where $\mathcal{O}(\Delta x)$ indicates that the order of accuracy of this approximation is $\sim \Delta x$. We say that this is *first order accurate*. This means that we are neglecting terms that scale as Δx. This is our truncation error (just as discussed above, arising because our numerical approximation throws away higher order terms). The approximation $\partial a/\partial x|_i = (a_{i+1} - a_i)/\Delta x$ has the same order of accuracy.

> **Exercise 1.3:** *Show that a centered difference, $\partial a/\partial x|_i = (a_{i+1} - a_{i-1})/(2\Delta x)$, is second order accurate, i.e. its truncation error is $\mathcal{O}(\Delta x^2)$.*

Figure 1.1 shows the left- and right-sided first-order differences and the central difference as approximations to $\sin(x)$. Generally speaking, higher-order methods have lower numerical error associated with them, and also involve a wider range of data points.

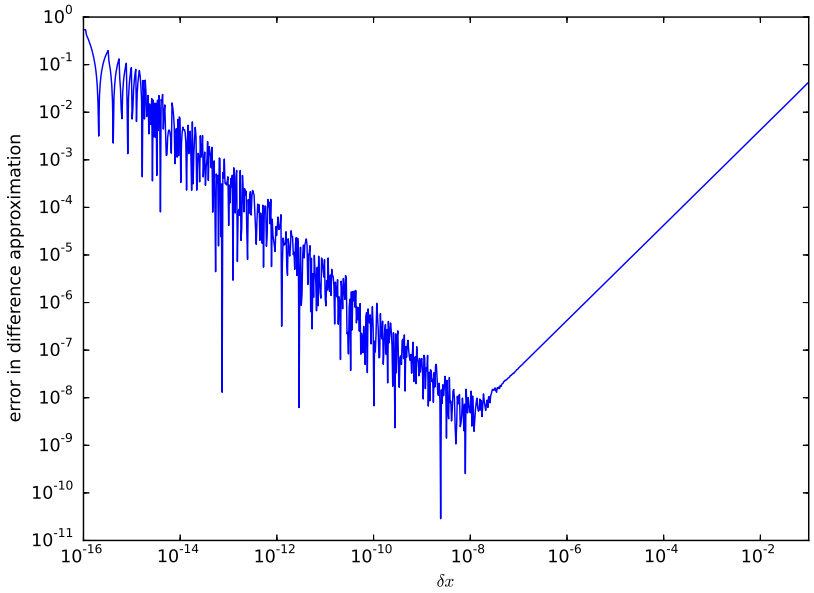

Figure 1.2: Error in the numerical approximation of the derivative of $f(x) = \sin(x)$ at $x_0 = 1$ as a function of the spacing δx. For small δx, roundoff error dominates the error in the approximation. For large δx, truncation error dominates.

An alternate scenario is when you know the analytic form of the function, $f(x)$, and are free to choose the points where you evaluate it. Here you can pick a δx and evaluate the derivative as

$$\left.\frac{df}{dx}\right|_{x=x_0} = \frac{f(x_0 + \delta x) - f(x_0)}{\delta x} \tag{1.8}$$

An optimal value for δx requires a balance of truncation error (which wants a small δx) and roundoff error (which becomes large when δx is close to machine ϵ). Figure 1.2 shows the error for the numerical derivative of $f(x) = \sin(x)$ at the point $x_0 = 1$, as a function of δx. A nice discussion of this is given in [54]. Comparing the result with different choices of δx allows for error estimation and an improvement of the result by combining the estimates using δx and $\delta x/2$ (this is the basis for a method called *Richardson extrapolation*).

Second- and higher-order derivatives can be constructed in the same fashion.

1.2.4 Integration

In numerical analysis, any integration method that is composed as a weighted sum of the function evaluated at discrete points is called a *quadrature rule*.

If we have a function sampled at a number of equally-spaced points, $x_0 \equiv a, x_1, \ldots, x_N \equiv$

b[1], we can construct a discrete approximation to an integral as:

$$I \equiv \int_a^b f(x)dx \approx \Delta x \sum_{i=0}^{N-1} f(x_i) \tag{1.9}$$

where $\Delta x \equiv (b-a)/N$ is the width of the intervals. This is a very crude method, but in the limit that $\Delta x \to 0$ (or $N \to \infty$), this will converge to the true integral. This method is called the *rectangle rule*. Note that here we expressing the integral over the N intervals using a simple quadrature rule in each interval. Summing together the results of the integral over each interval to get the result in our domain is called *compound* integration.

We can get a more accurate answer for I by interpolating between the points. The simplest case is to connect the sampled function values, $f(x_0), f(x_1), \ldots, f(x_N)$ with a line, creating a trapezoid in each interval, and then simply add up the area of all of the trapezoids:

$$I \equiv \int_a^b f(x)dx \approx \Delta x \sum_{i=0}^{N-1} \frac{f(x_i) + f(x_{i+1})}{2} \tag{1.10}$$

This is called the *trapezoid rule*. Note here we assume that the points are equally spaced.

One can keep going, but practically speaking, a quadratic interpolation is as high as one usually encounters. Fitting a quadratic polynomial requires three points.

> **Exercise 1.4:** *Consider a function, $f(x)$, sampled at three equally-spaced points, α, β, γ, with corresponding function values $f_\alpha, f_\beta, f_\gamma$. Derive the expression for Simpson's rule by fitting a quadratic $\hat{f}(x) = A(x-\alpha)^2 + B(x-\alpha) + C$ to the three points (this gives you A, B, and C), and then analytically integrating $\hat{f}(x)$ in the interval $[\alpha, \gamma]$. You should find*
>
> $$I = \frac{\gamma - \alpha}{6}(f_\alpha + 4f_\beta + f_\gamma) \tag{1.11}$$
>
> *Note that $(\gamma - \alpha)/6 = \Delta x/3$*

For a number of samples, N, in $[a, b]$, we will consider every two intervals together. The resulting expression is:

$$I \equiv \int_a^b f(x)dx \approx \frac{\Delta x}{3} \sum_{i=0}^{(N-2)/2} [f(x_{2i}) + 4f(x_{2i+1}) + f(x_{2i+2})] \tag{1.12}$$
$$= f(x_0) + 4f(x_1) + 2f(x_2) + 4f(x_3) + 2f(x_4) + \ldots + 2f(x_{N-2}) + 4f(x_{N-1}) + f(x_N)$$

[1] Note that this is N intervals and $N+1$ points

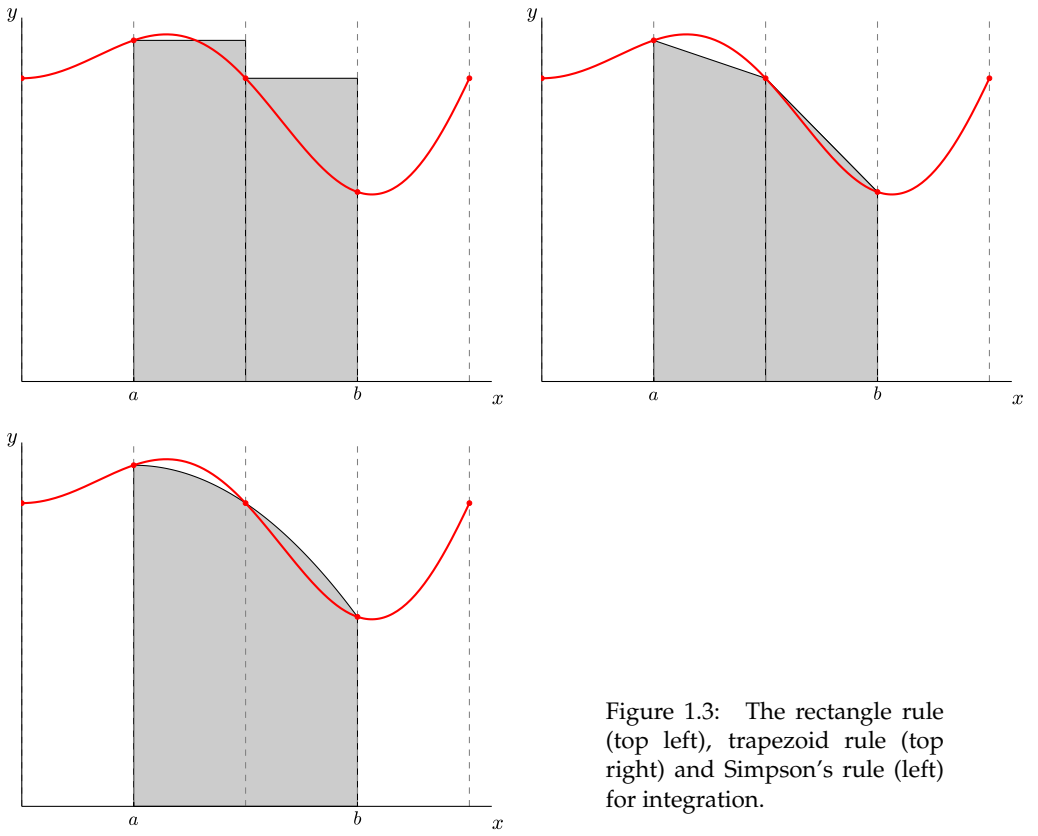

Figure 1.3: The rectangle rule (top left), trapezoid rule (top right) and Simpson's rule (left) for integration.

This method is called *Simpson's rule*. Note that for 2 intervals / 3 sample points ($N = 2$), we only have 1 term in the sum, $(N-2)/2 = 0$, and we get the result derived in Exercise 1.4.

Figure 1.3 shows these different approximations for the case of two intervals (three points).

Analogous expressions exist for the case of unequally-spaced points. The compound trapezoid rule converges as second-order over the interval $[a, b]$, while Simpson's rule converges as fourth-order.

As with differentiation, if you are free to pick the points where you evaluate $f(x)$, you can get a much higher-order accurate result. *Gaussian quadrature* is a very powerful technique that uses the zeros of different polynomials as the evaluation points for the function to give extremely accurate results. See the text by Garcia [27] for a nice introduction to these methods.

1.2.5 Root Finding

Often we want to find the root of a function (or the vector that zeros a vector of functions). The most popular method for root finding is the *Newton-Raphson*

method. This arises from a Taylor expansion. If you wish to find x, such that $f(x) = 0$, and start with an initial guess x_0 that you believe is close to the root, then you can improve the guess to the root by an amount δx as:

$$f(x_0 + \delta x) \sim f(x_0) + f'(x_0)\delta x + \ldots = 0 \tag{1.13}$$

Keeping only to $\mathcal{O}(\delta x)$,

$$\delta x = -\frac{f(x_0)}{f'(x_0)} \tag{1.14}$$

This can be used to correct out guess as $x_0 \leftarrow x_0 + \delta x$, and we can iterate on this procedure until δx falls below some tolerance.

The main 'gotya' with this technique is that you need a good initial guess. In the Taylor expansion, we threw away the δx^2 term, but if our guess was far from the root, this (and higher-orders) term may not be small. Obviously, the derivative must be non-zero in the region around the root that you search as well.

The *secant method* for root finding is essentially Newton-Raphson, but instead of using an analytic derivative, f' is estimated as a numerical difference.

Newton's method has pathologies—it is possible to get into a cycle where you don't converge but simply pass through the same set of root approximations. Many other root finding methods exist, including bisection, which iteratively halves an interval known to contain a root by looking for the change in sign of the function $f(x)$. Brendt's method combines several different methods to produce a robust procedure for root-finding.

1.2.6 ODEs

Consider a system

$$\dot{y}_k = f_k(t, \mathbf{y}(t)) \tag{1.15}$$

We want to solve for the vector $\mathbf{y}$ as a function of time. Broadly speaking, methods for integrating ODEs can be broken down into explicit and implicit methods. Explicit methods difference the system to form an update that uses only the current (and perhaps previous) values of the dependent variables in evaluating f. For example, a first-order explicit update (*Euler's method*) appears as:

$$y_k^{n+1} = y_k^n + \Delta t f_k(t^n, \mathbf{y}^n) \tag{1.16}$$

where Δt is the stepsize.

Perhaps the most popular explicit method for ODEs is the 4th-order Runge-Kutta method (RK4). This is a multistep method where several extrapolations to the midpoint and endpoint of the interval are made to estimate slopes and then a weighted average of these slopes are used to advance the solution. The various slopes are illustrated in Figure 1.5 and the overall procedure looks like:

$$y_k^{n+1} = y_k^n + \frac{\Delta t}{6}(k_1 + 2k_2 + 2k_3 + k_4) \tag{1.17}$$

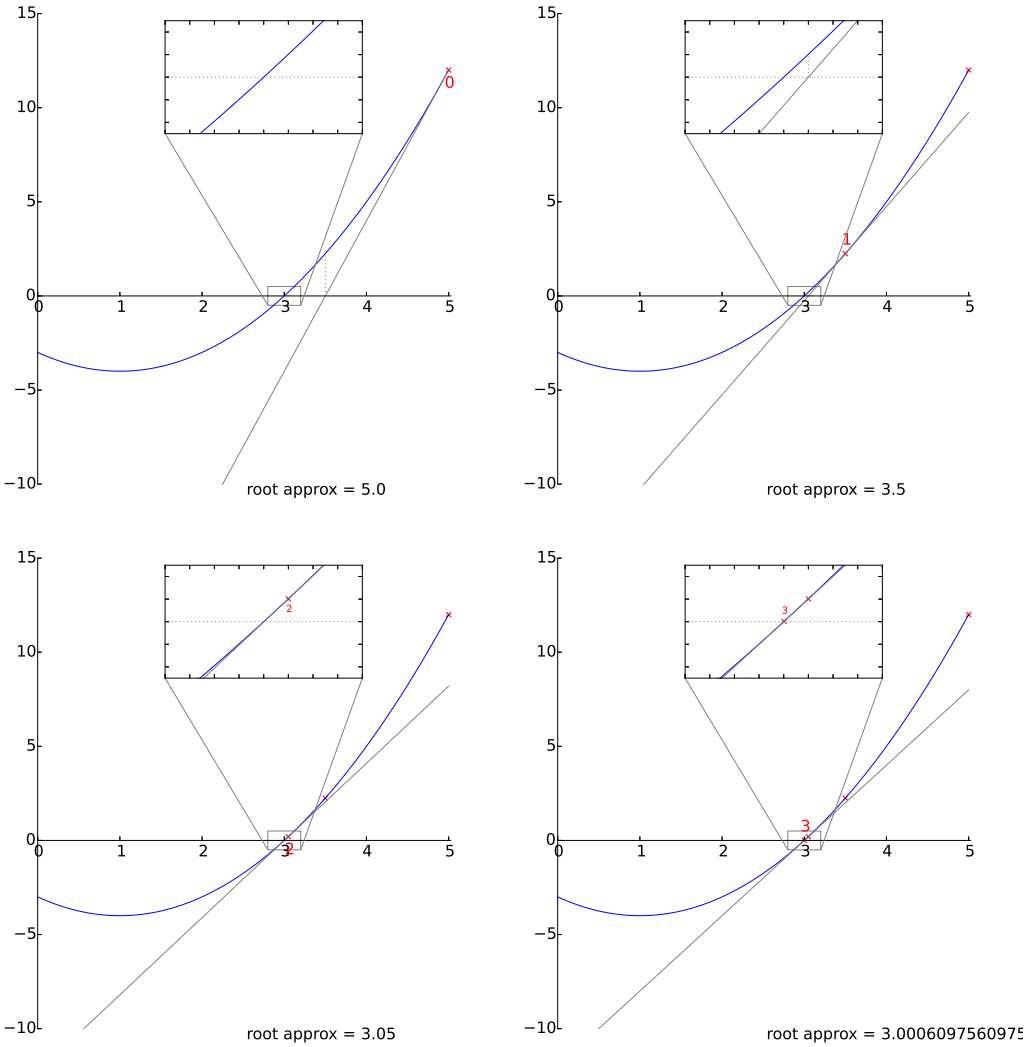

Figure 1.4: The convergence of Newton's method for finding the root. In each pane, the red point is the current guess for the root. The solid gray line is the extrapolation of the slope at the guess to the x-axis, which defines the next approximation to the root. The vertical dotted line to the function shows the new slope that will be used for extrapolation in the next iteration.

where the slopes are:

$$k_1 = f(t^n, y_k^n) \tag{1.18}$$
$$k_2 = f(t^n + \tfrac{\Delta t}{2}, y_k^n + \tfrac{\Delta t}{2} k_1) \tag{1.19}$$
$$k_3 = f(t^n + \tfrac{\Delta t}{2}, y_k^n + \tfrac{\Delta t}{2} k_2) \tag{1.20}$$
$$k_4 = f(t^n + \Delta t, y_k^n + \Delta t k_3) \tag{1.21}$$

Note the similarity to Simpson's method for integration. This is fourth-order accurate overall.

Exercise 1.5: *Consider the orbit of Earth around the Sun. If we work in the units of astronomical units, years, and solar masses, then Newton's gravitational constant and the solar mass together are simply $GM = 4\pi^2$. We can write the ODE system describing the motion of Earth as:*

$$\dot{\mathbf{x}} = \mathbf{v} \tag{1.22}$$

$$\dot{\mathbf{v}} = -\frac{GM\mathbf{r}}{r^3} \tag{1.23}$$

Integrate this system with the first-order Euler and the RK4 method and measure the convergence by integrating at a number of different Δts.

Implicit methods

Implicit methods difference the system in a way that includes the new value of the dependent variables in the evaluation of $f_k(t, \mathbf{y}(t))$—the resulting implicit system is usually solved using, for example, Newton-Raphson iteration.

A first-order implicit update (called *backward Euler*) is:

$$\mathbf{y}^{n+1} = \mathbf{y}^n + \Delta t \mathbf{f}(t^{n+1}, \mathbf{y}^{n+1}) \tag{1.24}$$

This is more complicated to solve than the explicit methods above, and generally will require some linear algebra. If we take Δt to be small, then the change in the solution, $\Delta\mathbf{y}$ will be small as well, and we can Taylor-expand the system.

To solve this, we pick a guess, $\mathbf{y}_0^{n+1}$, that we think is close, to the solution and we will solve for a correction, $\Delta\mathbf{y}_0$ such that

$$\mathbf{y}^{n+1} = \mathbf{y}_0^{n+1} + \Delta\mathbf{y}_0 \tag{1.25}$$

Using this approximation, we can expand the righthand side vector,

$$\mathbf{f}(t^{n+1}, \mathbf{y}^{n+1}) = \mathbf{f}(t^n, \mathbf{y}_0^{n+1}) + \left.\frac{\partial \mathbf{f}}{\partial \mathbf{y}}\right|_0 \Delta\mathbf{y}_0 + \ldots \tag{1.26}$$

Here we recognize the Jacobin matrix, $\mathbf{J} \equiv \partial\mathbf{f}/\partial\mathbf{y}$,

$$\mathbf{J} = \begin{pmatrix} \partial f_1/\partial y_1 & \partial f_1/\partial y_2 & \partial f_1/\partial y_3 & \ldots & \partial f_1/\partial y_n \\ \partial f_2/\partial y_1 & \partial f_2/\partial y_2 & \partial f_2/\partial y_3 & \ldots & \partial f_2/\partial y_n \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ \partial f_n/\partial y_1 & \partial f_n/\partial y_2 & \partial f_n/\partial y_3 & \ldots & \partial f_n/\partial y_n \end{pmatrix} \tag{1.27}$$

Putting this into Eq. 1.24, we have:

$$\mathbf{y}_0^{n+1} + \Delta\mathbf{y}_0 = \mathbf{y}^n + \Delta t \left[\mathbf{f}(t^{n+1}, \mathbf{y}_0^{n+1}) + \mathbf{J}|_0 \, \Delta\mathbf{y}_0\right] \tag{1.28}$$

Writing this as a system for the unknown correction, $\Delta\mathbf{y}_0$, we have

$$\left(\mathbf{I} - \Delta t\, \mathbf{J}|_0\right) \Delta\mathbf{y}_0 = \mathbf{y}^n - \mathbf{y}_0^{n+1} + \Delta t \mathbf{f}(t^{n+1}, \mathbf{y}_0^{n+1}) \tag{1.29}$$

This is a linear system (a matrix $\times$ vector $=$ vector) that can be solved using standard matrix techniques. After solving, we can correct our initial guess:

$$\mathbf{y}_1^{n+1} = \mathbf{y}_0^{n+1} + \Delta\mathbf{y}_0 \tag{1.30}$$

Written this way, we see that we can iterate. To kick things off, we need a suitable guess—a good choice is $\mathbf{y}_0^{n+1} = \mathbf{y}^n$. Then we correct this guess by iterating, with the k-th iteration looking like:

$$\left(\mathbf{I} - \Delta t\, \mathbf{J}|_{k-1}\right) \Delta\mathbf{y}_{k-1} = \mathbf{y}^n - \mathbf{y}_{k-1}^{n+1} + \Delta t \mathbf{f}(t^{n+1}, \mathbf{y}_{k-1}^{n+1}) \tag{1.31}$$

$$\mathbf{y}_k^{n+1} = \mathbf{y}_{k-1}^{n+1} + \Delta\mathbf{y}_{k-1} \tag{1.32}$$

We will iterate until we find $\|\Delta\mathbf{y}_k\| < \epsilon\|\mathbf{y}^n\|$. Here ϵ is a small tolerance, and we use $\mathbf{y}^n$ to produce a reference scale for meaningful comparison. Note that here we use a vector norm to give a single number for comparison.

Note that the role of the Jacobian here is the same as the first derivative in the scalar Newton's method for root finding (Eq. 1.14)—it points from the current guess to the solution. Sometimes an approximation to the Jacobian, which is cheaper to evaluate, may work well enough for the method to converge.

Explicit methods are easier to program and run faster (for a given Δt), but implicit methods work better for *stiff* problems—those characterized by widely disparate timescales over which the solution changes [16][2]. A good example of this issue in astrophysical problems is with nuclear reaction networks (see, e.g., [50]). As with the explicit case, higher-order methods exist that can provide better accuracy at reduced cost.

1.2.7 FFTs

[2]Defining whether a problem is stiff can be tricky. For a system of ODEs, a large range in the eigenvalues of the Jacobian usually means it is stiff.

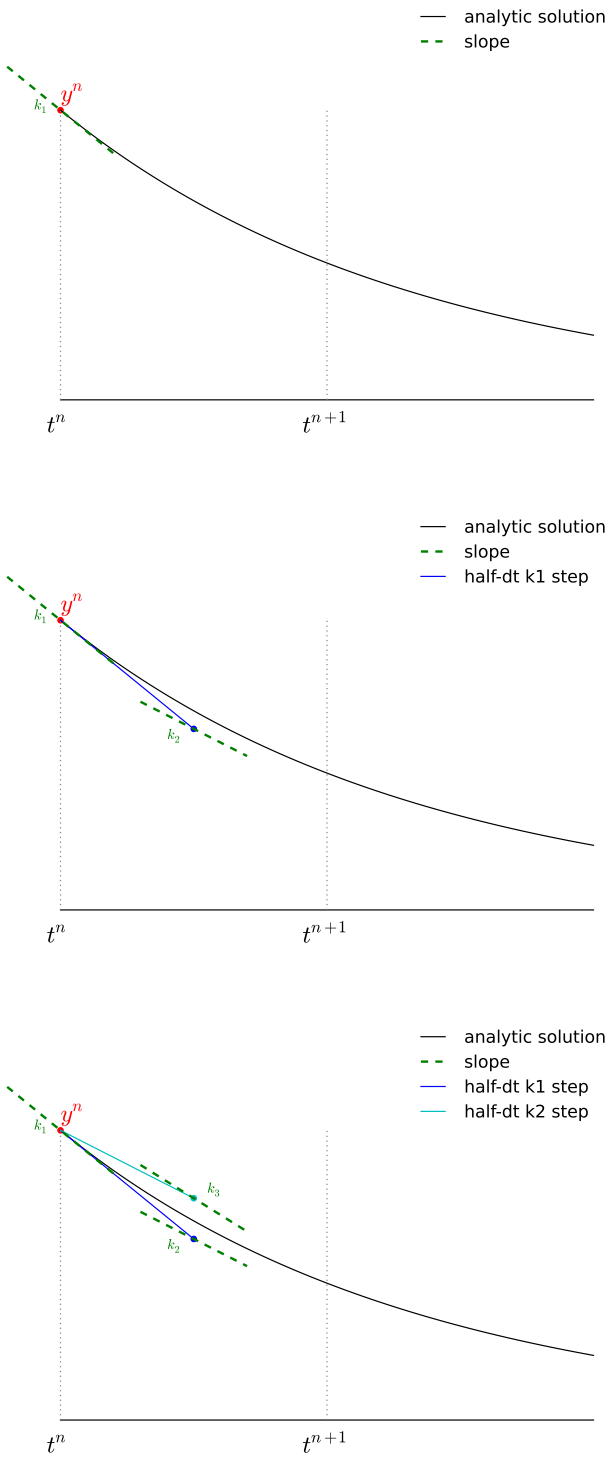

Figure 1.5: A graphical illustration of the four steps in the 4$^{\text{th}}$-order Runge-Kutta method. Annotate this better

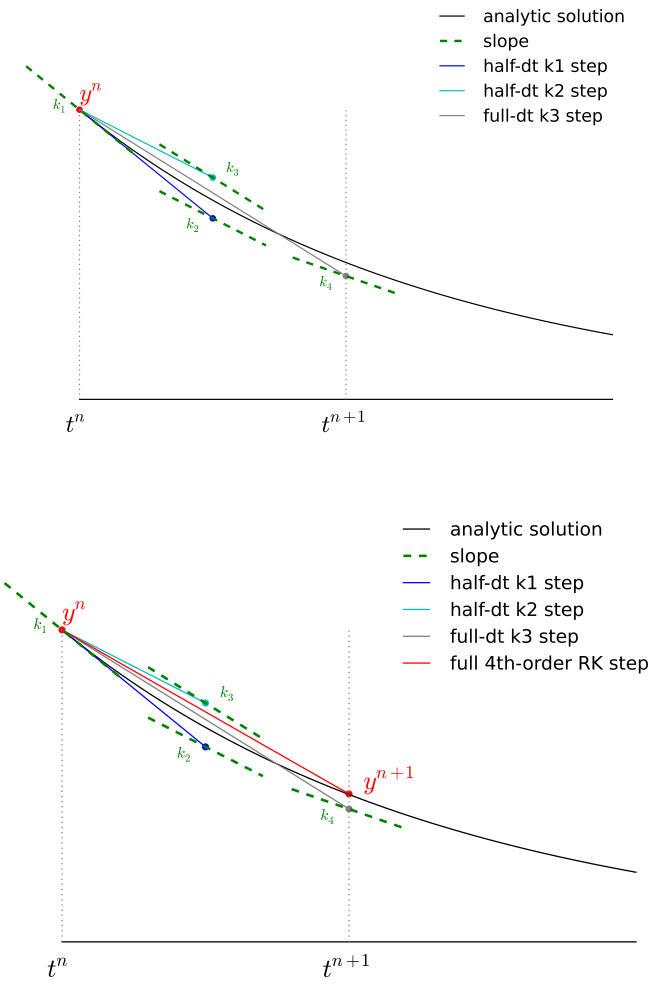

Figure 1.5: (continued) A graphical illustration of the four steps in the 4$^{\text{th}}$-order Runge-Kutta method.

Chapter 2

Finite-Volume Grids

2.1 Discretization

The physical systems we model are described by continuous mathematical functions, $f(x, t)$ and their derivatives in space and time. To represent this continuous system on a computer we must discretize it—convert the continuous function into a discrete number of points in space at one or more discrete instances in time. There are many different discretization methods used throughout the physical sciences, engineering, and applied mathematics fields, each with their own strengths and weaknesses. Broadly speaking, we can divide these methods into grid-based and gridless methods.

Gridless methods include those which represent the function as a superposition of continuous basis functions (e.g. sines and cosines). This is the fundamental idea behind *spectral methods*. A different class of methods are those that use discrete particles to represent the mass distribution and produce continuous functions by integrating over these particles with a suitable kernel—this is the basis of *smoothed particle hydrodynamics* (SPH) [39]. SPH is a very popular method in astrophysics.

For grid-based methods, we talk about both the style of the grid (structured vs. unstructured) and the discretization method, e.g. the finite-difference, finite-volume, and finite-element methods.

Structured grids are logically Cartesian. This means that you can reference the location of any cell in the computational domain via an integer index in each spatial dimension. From a programming standpoint, the grid structure can be represented exactly by a multi-dimensional array. Unstructured grids don't have this simple pattern. A popular type of unstructured grid is created using triangular cells (in 2-d) or tetrahedra (in 3-d). The main advantage of these grids is that you can easily represent irregularly-shaped domains. The disadvantage is that the data structures required to describe the grid are more complicated than a simple array

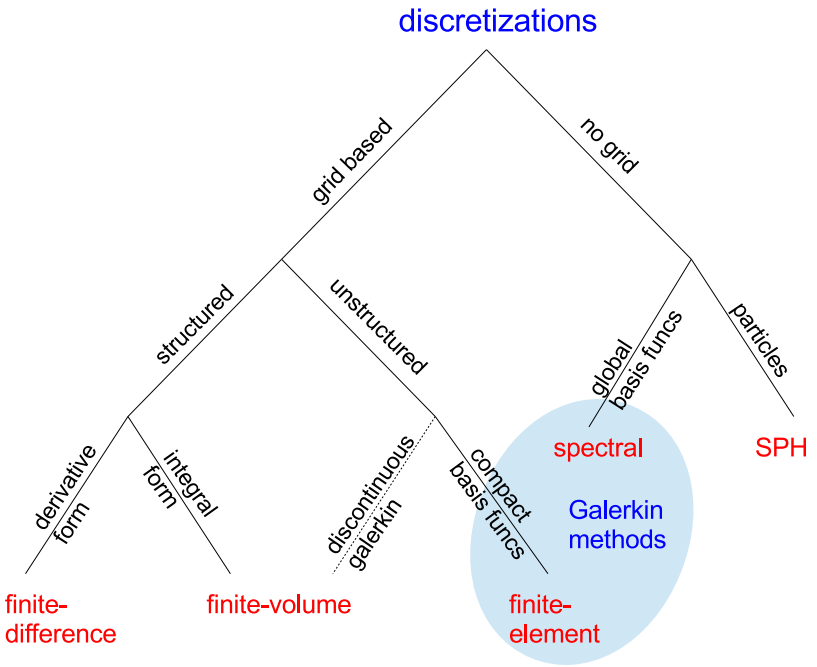

Figure 2.1: Taxonomy of different discretizations.

(and tend to have more inefficient memory access).

Once a grid is established, the system of PDEs is converted into a system of discrete equations on the grid. Finite element methods are widely used in engineering together with unstructured grids. A type of finite-element method called a *Galerkin method* can be formed by integrating the continuous function against a compact basis set represented by tetrahedra. These methods work well with the irregular geometries that arise in engineering. Finite-difference and finite-volume methods can both be applied to *structured grids*. The main difference between these methods is that the finite-difference methods build from the differential form of PDEs while the finite-volume methods build from the integral form of the PDEs. Both of these methods find wide use in astrophysics.

In these notes, we will focus on finite-volume techniques on structured grids.

2.2 Grid basics

The grid is the fundamental object for representing continuous functions in a discretized fashion, making them amenable to computation. In astrophysics, we typi-

cally use structured grids—these are logically Cartesian, meaning that the position of a quantity on the grid can be specified by a single integer index in each dimension.

We represent derivatives numerically by discretizing the domain into a finite number of zones (a numerical grid). This converts a continuous derivative into a difference of discrete data. Different approximations have different levels of accuracy.

There are two main types of structured grids used in astrophysics: *finite-difference* and *finite-volume*. These differ in way the data is represented. On a finite-difference grid, the discrete data is associated with a specific point in space. On a finite-volume grid, the discrete data is represented by averages over a control volume. Nevertheless, these methods can often lead to very similar looking discrete equations.

Consider the set of grids show in Figure 2.2. On the top is a classic finite-difference grid. The discrete data, f_i, are stored as points regularly spaced in x. With this discretization, the spatial locations of the points are simply $x_i = i\Delta x$, where $i = 0, \ldots, N-1$. Note that for a finite-sized domain, we would put a grid point precisely on the physical boundary at each end.

The middle grid is also finite-difference, but now we imagine first dividing the domain into N cells or zones, and we store the discrete data, f_i, at the center of the zone. This is often called a *cell-centered finite-difference* grid. The physical coordinate of the zone centers (where the data lives) are: $x_i = (i + 1/2)\Delta x$, where $i = 0, \ldots N-1$. Note that now for a finite-sized domain, the left edge of the first cell will be on the boundary and the first data value will be associated at a point $\Delta x/2$ inside the boundary. A similar situation arises at the right physical boundary.

Finally, the bottom grid is a finite-volume grid. The layout looks identical to the cell-centered finite difference grid, except now instead of the discrete data being associated at a single point in space, we draw a line to show that it is taken as the average in a zone. We label it as $\langle f \rangle_i$ to indicate the average. We label the left and right edges of a zone with half-integer indices $i - 1/2$ and $i + 1/2$. The physical coordinate of the center of the zone is the same as in the cell-centered finite-difference case.

In all cases, for a *regular* structured grid, we take Δx to be constant. For the finite-difference grids, the discrete value at each point is obtains from the continuous function $f(x)$ as:

$$f_i = f(x_i) \tag{2.1}$$

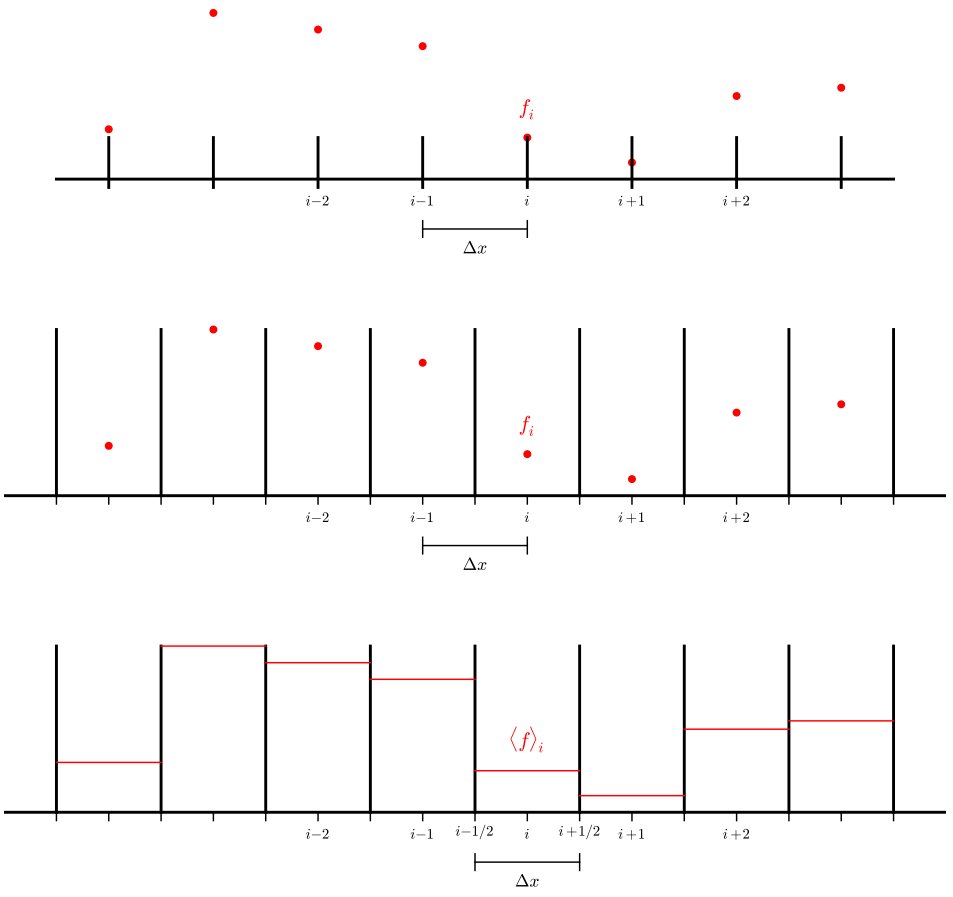

Figure 2.2: Different types of structured grids. Top: a finite-difference grid—the discrete data are associated with a specific point in space. Middle: a cell-centered finite-difference grid—again the data is at a specific point, but now we imagine the domain divided into zone with the data living at the center of each zone. Bottom: a finite-volume grid—here the domain is divided into zones and we store the average value of the function within each zone.

2.3 Finite-volume grids

In the finite-volume discretization, f_i represents the average of $f(x,t)$ over the interval $x_{i-1/2}$ to $x_{i+1/2}$, where the half-integer indices denote the zone edges (i.e. $x_{i-1/2} = x_i - \Delta x/2$):

$$\langle f \rangle_i = \frac{1}{\Delta x} \int_{x_{i-1/2}}^{x_{i+1/2}} f(x) dx \tag{2.2}$$

The lower panel of Figure 2.2 shows a finite-volume grid, with the half-integer indices bounding zone i marked. Here we've drawn $\langle a \rangle_i$ as a horizontal line spanning the entire zone—this is to represent that it is an average within the volume

defined by the zone edges. To second-order accuracy,

$$\langle f \rangle_i = \frac{1}{\Delta x} \int_{x_{i-1/2}}^{x_{i+1/2}} f(x) dx \sim f(x_i) \tag{2.3}$$

This means that we can treat the average of f over a zone as simply $f(x)$ evaluated at the zone center if we only want second-order accuracy.

Exercise 2.1: *Show that Eq. 2.3 is true to $O(\Delta x^2)$ by expanding $f(x)$ as a Taylor series in the integral.*

Exercise 2.2: *A conservative interpolant is used to reconstruct a continuous functional form, $f(x)$, from a collection of cell-averages. A key requirement is that when $f(x)$ is averaged over a cell, it returns the cell-average.*

Consider three cell averages: $\langle f \rangle_{i-1}$, $\langle f \rangle_i$, $\langle f \rangle_{i+1}$. Fit a quadratic polynomial through these points,

$$f(x) = A(x - x_i)^2 + B(x - x_i) + C \tag{2.4}$$

where the coefficients, A, B, and C are found by applying the constraints:

$$\langle f \rangle_{i-1} = \frac{1}{\Delta x} \int_{x_{i-3/2}}^{x_{i-1/2}} a(x) dx \tag{2.5}$$

$$\langle f \rangle_i = \frac{1}{\Delta x} \int_{x_{i-1/2}}^{x_{i+1/2}} a(x) dx \tag{2.6}$$

$$\langle f \rangle_{i+1} = \frac{1}{\Delta x} \int_{x_{i+1/2}}^{x_{i+3/2}} a(x) dx \tag{2.7}$$

Show that the conservative interpolant is:

$$\begin{aligned} f(x) = &\frac{\langle f \rangle_{i-1} - 2\langle f \rangle_i + \langle f \rangle_{i+1}}{2\Delta x^2}(x - x_i)^2 + \\ &\frac{\langle f \rangle_{i+1} - \langle f \rangle_{i-1}}{2\Delta x}(x - x_i) + \\ &\frac{-\langle f \rangle_{i-1} + 26\langle f \rangle_i - \langle f \rangle_{i+1}}{24} \end{aligned} \tag{2.8}$$

The IPython notebook hydro_examples: `conservative-interpolation.ipynb` shows how to derive these interpolants using SymPy.

2.3.1 Differences and order of accuracy

In practice, when computing derivatives in a finite-volume discretization, we can use the second-order centered difference from § 1.2.3 treating the finite-volume data as living at the cell-centers and still be second-order accurate. For higher accuracy, we can fit a *conservative interpolant* (as illustrated in exercise 2.2) to a collection of points and then differentiate the interpolant itself.

Notice that the righthand side of all derivative approximations involve some linear combinations of a_i's. We call this the *stencil*. The *width* of the stencil is a measure of how many zones on either side of zone i we reach when computing our approximation.

2.3.2 Conservation

The finite-volume grid is useful when dealing with conservation laws. Consider the following system:

$$\frac{\partial U}{\partial t} + \nabla \cdot F(U) = 0 \tag{2.9}$$

where U is a vector of conserved quantities and $F(U)$ is the flux of each quantity. This type of system arises, for example, in fluid flow, where the system will represent conservation of mass, momentum, and energy.

Consider one-dimension. Integrating this system over a zone, and normalizing by Δx, we have:

$$\frac{\partial \langle U \rangle_i}{\partial t} = -\frac{1}{\Delta x} \int_{x_{i-1/2}}^{x_{i+1/2}} \frac{\partial F}{\partial x} dx = -\frac{1}{\Delta x} \left\{ F(U)|_{x_{i+1/2}} - F(U)|_{x_{i-1/2}} \right\} \tag{2.10}$$

Independent of how we discretize in time, notice that we have the cell-average on the left and that the righthand side is simply a difference of fluxes through the interfaces of the zone. For the $i+1$ zone, the update would be:

$$\frac{\partial \langle U \rangle_{i+1}}{\partial t} = -\frac{1}{\Delta x} \left\{ F(U)|_{x_{i+3/2}} - F(U)|_{x_{i+1/2}} \right\} \tag{2.11}$$

Notice that this shares the flux at the $x_{i+1/2}$ interface, but with the opposite sign. When all of the updates are done, the flux through each boundary adds to one zone and subtracts from its neighbor, exactly conserving (to round-off error) the quantity U. This cancellation of the sums is an example of a *telescoping property*, and is the main reason why finite-volume methods are attractive—conserved quantities are conserved to machine (roundoff) precision.

Note that conservation is not the same as accuracy. We can construct the fluxes for Eq. 2.11 by calling a random number generator and we'd still be conservative, but not at all accurate.

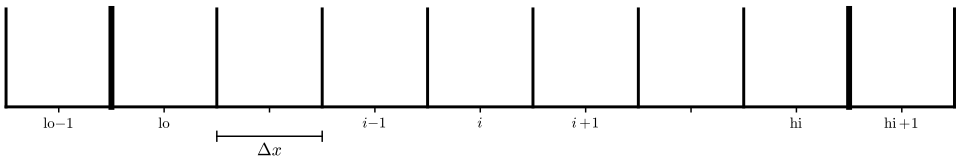

Figure 2.3: A simple 1-d finite-volume grid with a single ghost cell at each end of the domain. The domain boundaries are indicated by the heavy vertical lines. Here there are $\mathrm{hi} - \mathrm{lo} + 1$ zones used in the discretization of the domain, with the first zone in the domain labeled lo and the last in the domain labeled hi.

2.3.3 Boundary conditions with finite-volume grids

Imagine that we wish to compute the derivative in every zone using:

$$\left.\frac{\partial a}{\partial x}\right|_i = \frac{a_i - a_{i-1}}{\Delta x} \quad . \tag{2.12}$$

If we denote the index corresponding to the leftmost zone in our domain as 'lo', then when we try to compute $\partial a/\partial x|_{\mathrm{lo}}$, we will "fall-off" the grid, i.e., we need a value of a for zone $\mathrm{lo} - 1$, which is outside the domain. This is where boundary conditions for our grid come into play.

We implement boundary conditions by extending the computational grid beyond the physical domain (see Figure 2.3). These additional zones are called *ghost cells*. They exist solely to handle the boundary conditions and allow us to use the same update equation (e.g. Eq. 2.12) for all zones in the domain.

The number of ghostcells needed for the grid depends on how wide the stencils used in the computation are. The wider the stencil, the more ghostcells are needed.

Periodic boundary conditions would be implemented as:

$$a_{\mathrm{hi}+1} = a_{\mathrm{lo}} \tag{2.13}$$
$$a_{\mathrm{lo}-1} = a_{\mathrm{hi}} \tag{2.14}$$

A simple outflow (zero-gradient) boundary condition would be implemented as:

$$a_{\mathrm{hi}+1} = a_{\mathrm{hi}} \tag{2.15}$$
$$a_{\mathrm{lo}-1} = a_{\mathrm{lo}} \tag{2.16}$$

2.4 Going further

- *Domain decomposition*: when running on a parallel computer, the work is divided up across processor using domain decomposition. Here, we break

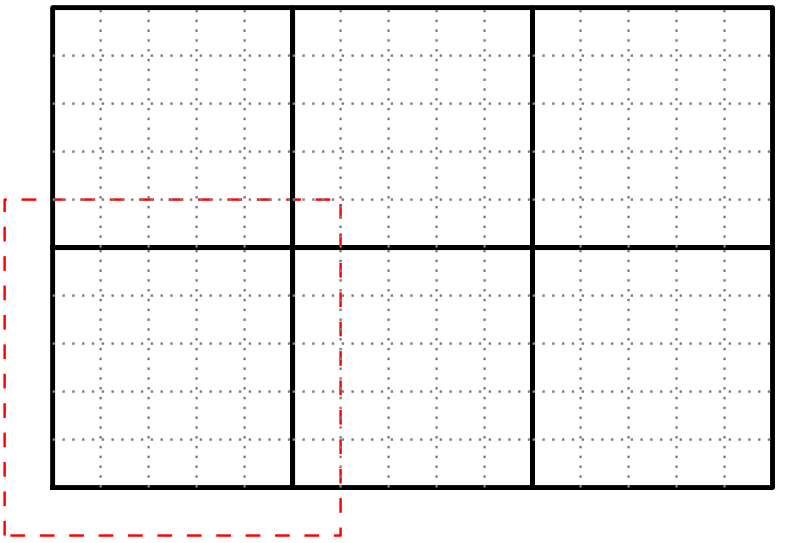

Figure 2.4: Domain decomposition of the computational domain into 6 separate sub-domains. Each sub-domain here has 5×5 zones. For one of the sub-domains, the perimeter of ghost cells is illustrated as the red boundary.

the computational domain into smaller sub-domains, and put one (or more) sub-domains on each processor. Each sub-domain has its own perimeter of ghost cells. These are now filled by copying information from the neighboring sub-domains or using the physical boundary conditions for the full domain, depending on where the ghost cells lie. Figure 2.4 shows a simple decomposition of a domain into 6 sub-domains.

- *AMR for structured grids*: adaptive mesh refinement uses a hierarchy of grids to focus resolution in regions of complex flow. For finite-volume codes, the standard reference for AMR is Berger & Colella [13]. Each level is an even integer multiple finer in resolution, and the grid cells line up with one another (i.e. in two-dimensions, four fine cells will be completely enclosed by a single coarse cell, when using a jump in resolution of $2\times$.) This provides a natural way to enforce conservation. At coarse-fine interfaces, corrections are done to ensure consistency.
- *Voronoi grids*: a hybrid of particle and grid methods is provided by methods that move particles in a Lagrangian fashion and use a Voronoi tessellation of these particles to define the grid that finite-volume methods are applied to. See, for example, the Arepo code paper [47].

Chapter 3

Advection

3.1 The Linear Advection Equation

The linear advection equation is simply:

$$a_t + ua_x = 0 \tag{3.1}$$

where $a(x,t)$ is some scalar quantity and u is the velocity at which it is advected ($u > 0$ advects to the right). The solution to Eq. 3.1 is to simply take the initial data, $a(x, t = 0)$, and displace it to the right at a speed u. The shape of the initial data is preserved in the advection. Many hyperbolic systems of PDEs, e.g. the equations of hydrodynamics, can be written in a form that looks like a system of (nonlinear) advection equations, so the advection equation provides important insight into the methods used for these systems.

> **Exercise 3.1:** *Show that $a(x - ut)$ is a solution to Eq. 3.1 for any choice of a. This means that the solution is constant along the lines $x = ut$ (the curves along which the solution is constant are called the characteristics).*

3.2 First-order advection in 1-d and finite-differences

To get a flavor of the methods for advection, we will use a simple finite-difference discretization—here, the domain is divided into a sequence of points where we store the solution. We will solve Eq. 3.1 numerically by discretizing the solution at these points. The index i denotes the point's location, and a_i denotes the discrete value of $a(x)$ in zone i. The data in each zone can be initialized as $a_i = a(x_i)$. Figure 3.1 shows the grid.

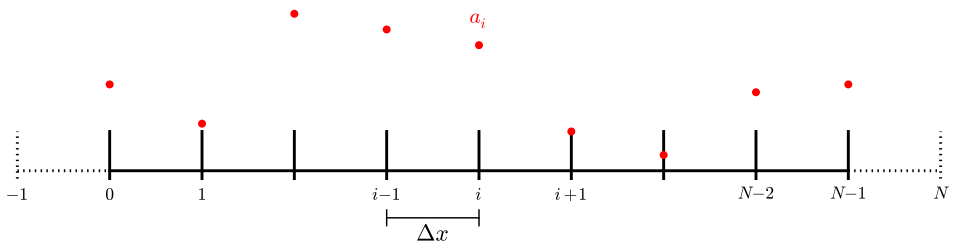

Figure 3.1: A simple finite-difference grid. The solution is stored at each of the labeled points. The dotted lines show the ghost points used to extend our grid past the physical boundaries to accommodate boundary conditions. Note that if we are periodic, then points 0 and $N-1$ are at the same physical point in space, so we would only need to update one of them.

We also need to discretize in time. We denote the time-level of the solution with a superscript, so $a_i^n = a(x_i, t^n)$. For a fixed Δt, time level n corresponds to a time of $t = n\Delta t$.

A simple first-order accurate discretization is:

$$\frac{a_i^{n+1} - a_i^n}{\Delta t} = -u \frac{a_i^n - a_{i-1}^n}{\Delta x} \tag{3.2}$$

This is an *explicit* method, since the new solution, a_i^{n+1}, depends only on information at the old time level, n.

Finally, we also need to specify a boundary condition for this. Our choice of spatial derivative is one-sided—it uses information from the zone to the left of the zone we are updating. This is because information is flowing from left to right in this problem ($u > 0$). This choice of the derivative is called *upwinding*—this choice of derivative results in a stable method. Notice that if we use Eq. 3.2 to update the data in the first zone inside the boundary, we need data to the left of this zone—outside of the domain. This means that we need a single ghost point to implement the boundary conditions for our method. The presence of the ghost points allow us to use the same update equation (e.g. Eq. 3.2) for all zones in the domain.

The last piece of information needed to update the solution is the timestep, Δt. It can be shown that for the solution to be *stable*, the timestep must be less than the time it takes information to propagate across a single zone. That is:

$$\Delta t \leq \frac{\Delta x}{u} \quad . \tag{3.3}$$

This is called the *Courant-Friedrichs-Lewy* or *CFL* condition. A dimensionless quantity called the *CFL number* is defined as

$$C = \frac{\Delta t u}{\Delta x} \tag{3.4}$$

Stability requires $C \leq 1$. We traditionally write the timestep as

$$\Delta t = C\frac{\Delta x}{u} \tag{3.5}$$

and specify C as part of the problem (a typical value may be $C = 0.7$).

Exercise 3.2: *Show analytically that when you use $C = 1$ in the first-order differenced advection equation (Eq. 3.2) that you advect the profile exactly, without any numerical error.*

Keep in mind that, in general, we will be solving a non-linear system of equations, so it is not possible to run with $C = 1$, since u (and therefore C) will change from zone to zone. Instead, one looks at the most restrictive timestep over all the zones and uses that for the entire system.

Exercise 3.3: *Write a code to solve the 1-d linear advection equation using the discretization of Eq. 3.2 on the domain $[0, 1]$ with $u = 1$ and periodic boundary conditions. For initial conditions, try both a Gaussian profile and a top-hat:*

$$a = \begin{cases} 0 & \text{if} \quad x < 1/3 \\ 1 & \text{if} \quad 1/3 \leq x < 2/3 \\ 0 & \text{if} \quad 2/3 \leq x \end{cases} \tag{3.6}$$

Note: For a general treatment of boundary conditions, you would initialize the ghost points to their corresponding periodic data and apply the difference equations to zones $0, \ldots, N-1$. However, for periodic BCs on this grid, points 0 and $N-1$ are identical, so you could do the update in this special case on points $1, \ldots, N-1$ without the need for ghost points and then set $a_0 = a_{N-1}$ after the update.

Run you program for one or more periods (one period is $T = 1/u$) with several different CFL numbers and notice that there is substantial numerical dissipation (see Figure 3.2).

Exercise 3.4: *You may think that using a centered-difference for the spatial derivative, $u_x \sim (u_{i+1} - u_{i-1})/(2\Delta x)$ would be more accurate. This method is called FTCS (forward-time, centered-space). Try this. You will find that the solution is unconditionally unstable.*

The classic method for understanding stability is to consider the growth of a single Fourier mode in our discretization. That is, substitute in $a_i^n = A^n e^{ji\theta}$, where $j =$

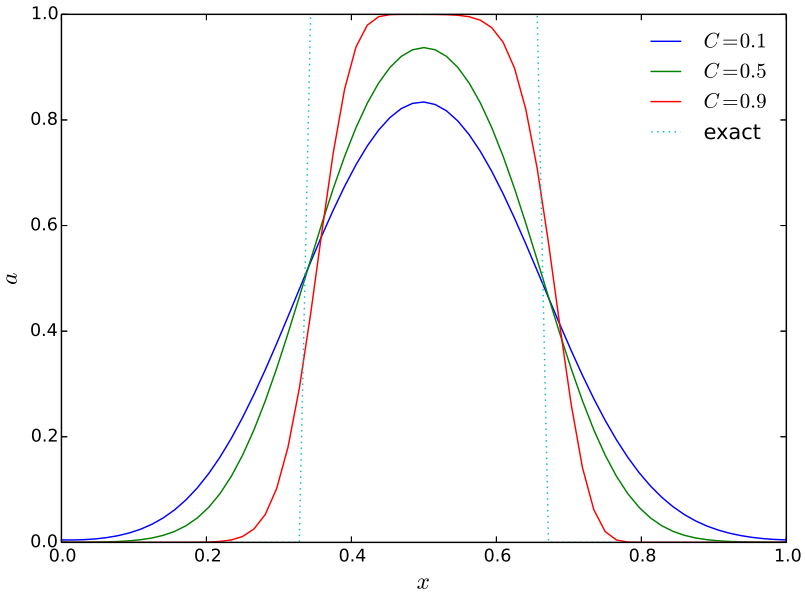

Figure 3.2: Finite-difference solution to the first-order finite-difference upwind method for advection, using 65 points and a variety of CFL numbers.
hydro_examples: `fdadvect.py`

$\sqrt{-1}$, and θ represents a phase. A method is stable if $|A^{n+1}/A^n| \leq 1$. Performing this with FTCS shows that no value of C can make the method stable. Doing the same analysis for Eq. 3.2 would show that the upwind method is stable for $0 \leq C \leq 1$. (Note that this stability analysis only works for linear equations, where the difference Fourier modes are decoupled, nevertheless, we use its ideas for nonlinear advection problems as well).

Exercise 3.5: *To get an alternate feel for stability, we can ask what the terms left out by truncation look like. That is, we can begin with the discretized equation:*

$$a_i^{n+1} - a_i^n = -\frac{u\Delta t}{\Delta x}(a_i^n - a_{i-1}^n) \tag{3.7}$$

and replace a_i^{n+1} with a Taylor expansion in time, and replace a_{i-1}^n with a Taylor expansion in space, keeping terms to $O(\Delta t^3)$ and $O(\Delta x^3)$. Replacing $\partial a/\partial t$ with $-u\partial a/\partial x$ in the higher-order terms, show that our difference equation more closely corresponds to

$$a_t + ua_x = \frac{u\Delta x}{2}\left(1 - \frac{\Delta t u}{\Delta x}\right)\frac{\partial^2 a}{\partial x^2} \tag{3.8}$$

$$= \frac{u\Delta x}{2}(1 - C)\frac{\partial^2 a}{\partial x^2} \tag{3.9}$$

Notice that the righthand side looks like a diffusion term, however, if $C >$

1, *then the coefficient of the diffusion is negative—this is unphysical. This means that the diffusion would act to take smooth features and make them more strongly peaked—the opposite of physical diffusion.*

An alternate approach to time-discretization is to do an implicit discretization. Here our upwind method would appear as:

$$\frac{a_i^{n+1} - a_i^n}{\Delta t} = -u \frac{a_i^{n+1} - a_{i-1}^{n+1}}{\Delta x} \tag{3.10}$$

We can write this as a linear system with coupled equations:

$$-Ca_{i-1}^{n+1} + (1+C)a_i^{n+1} = a_i^n \tag{3.11}$$

In matrix form, solving for the points $1, \ldots, N-1$, this is:

$$\begin{pmatrix} 1+C & & & & & & -C \\ -C & 1+C & & & & & \\ & -C & 1+C & & & & \\ & & -C & 1+C & & & \\ & & & \ddots & \ddots & & \\ & & & & -C & 1+C & \\ & & & & & -C & 1+C \end{pmatrix} \begin{pmatrix} u_1^{n+1} \\ u_2^{n+1} \\ u_3^{n+1} \\ u_4^{n+1} \\ \vdots \\ u_{N-2}^{n+1} \\ u_{N-1}^{n+1} \end{pmatrix} = \begin{pmatrix} u_1^n \\ u_2^n \\ u_3^n \\ u_4^n \\ \vdots \\ u_{N-2}^n \\ u_{N-1}^n \end{pmatrix} \tag{3.12}$$

This requires a matrix solve—this makes implicit methods generally more expensive than explicit methods. However, stability analysis would show that this implicit discretization is stable for any choice of C. (But one must not confuse stability with accuracy—the most accurate solutions with this method will still have a small C). Also note that the form of the matrix will change depending on the choice of boundary conditions. Figure 3.3 shows the result of solving this implicit system.

3.3 Second-order advection in 1-d and the finite-volume method

In these notes, we will typically use a *finite-volume* discretization. Here we explore this method for the advection equation. First we rewrite the advection equation in *conservation form*:

$$a_t + [f(a)]_x = 0 \tag{3.13}$$

where $f(a) = ua$ is the flux of the quantity a. In conservation form, the time derivative of a quantity is related to the divergence of its flux.

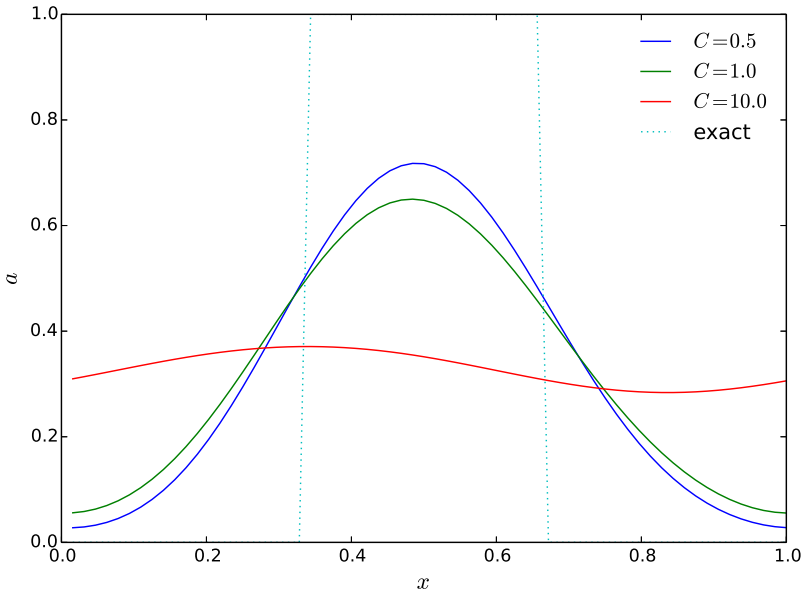

Figure 3.3: Finite-difference solution to the implicit first-order finite-difference upwind method for advection, using 65 points and a variety of CFL numbers.
hydro_examples: `fdadvect_implicit.py`

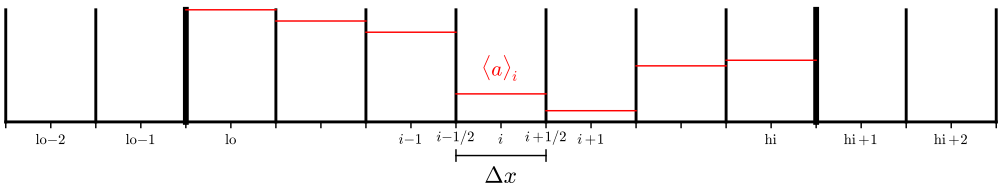

Figure 3.4: A finite-volume grid running from lo,...,hi, with two ghost cells at each end.

Recall that in the finite-volume discretization, $\langle a \rangle_i$ represents the average of $a(x,t)$ over the interval $x_{i-1/2}$ to $x_{i+1/2}$, where the half-integer indexes denote the zone edges (i.e. $x_{i-1/2} = x_i - \Delta x/2$). Figure 3.4 shows an example of such a grid with 2 ghost cells at each end. (For simplicity of notation, we drop the $\langle\rangle$ going forward). To discretize Eq. 3.13, we integrate it over a zone, from $x_{i-1/2}$ to $x_{i+1/2}$, normalizing by the zone width, Δx:

$$\frac{1}{\Delta x}\int_{x_{i-1/2}}^{x_{i+1/2}} a_t\, dx = -\frac{1}{\Delta x}\int_{x_{i-1/2}}^{x_{i+1/2}} \frac{\partial}{\partial x} f(a)\, dx \tag{3.14}$$

$$\frac{\partial}{\partial t} a_i = -\frac{1}{\Delta x}\left\{ [f(a)]_{i+1/2} - [f(a)]_{i-1/2} \right\} \tag{3.15}$$

This is an evolution equation for the zone-average of a, and shows that it updates in time based on the fluxes through the boundary of the zone. We discretize in time

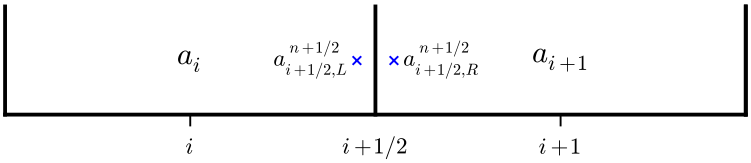

Figure 3.5: The left and right interface state at the $i + 1/2$ interface. Here, the left state, $a^{n+1/2}_{i+1/2,L}$, was predicted to the interface from the zone to the left of the interface, using a_i, and the right state, $a^{n+1/2}_{i+1/2,R}$, was predicted to the interface from the zone to the right, using a_{i+1}.

by evaluating the righthand side at the midpoint in time—this gives a centered difference in time, which is second-order accurate:

$$\frac{a_i^{n+1} - a_i^n}{\Delta t} = -\frac{[f(a)]^{n+1/2}_{i+1/2} - [f(a)]^{n+1/2}_{i-1/2}}{\Delta x} \tag{3.16}$$

To evaluate the fluxes at the half-time, we need the state at the half-time, that is, we do :

$$[f(a)]^{n+1/2}_{i+1/2} = f(a^{n+1/2}_{i+1/2}) \ . \tag{3.17}$$

We construct a second-order accurate approximation to $a^{n+1/2}_{i+1/2}$ by Taylor expanding the data in the cell to the interface. Note that for each interface, there are two possible interface states we can construct—one using the data to the left of the interface (which we will denote with a "L" subscript) and the other using the data to the right of the interface (denoted with an "R" subscript)—see Figure 3.5. These states are:

$$\begin{aligned} a^{n+1/2}_{i+1/2,L} &= a_i^n + \frac{\Delta x}{2}\left.\frac{\partial a}{\partial x}\right|_i + \frac{\Delta t}{2}\left.\frac{\partial a}{\partial t}\right|_i + \mathcal{O}(\Delta x^2) + \mathcal{O}(\Delta t^2) \\ &= a_i^n + \frac{\Delta x}{2}\left.\frac{\partial a}{\partial x}\right|_i + \frac{\Delta t}{2}\left(-u\left.\frac{\partial a}{\partial x}\right|_i\right) + \ldots \\ &= a_i^n + \frac{\Delta x}{2}\left(1 - \frac{\Delta t}{\Delta x}u\right)\left.\frac{\partial a}{\partial x}\right|_i + \ldots \end{aligned} \tag{3.18}$$

$$\begin{aligned} a^{n+1/2}_{i+1/2,R} &= a_{i+1}^n - \frac{\Delta x}{2}\left.\frac{\partial a}{\partial x}\right|_{i+1} + \frac{\Delta t}{2}\left.\frac{\partial a}{\partial t}\right|_{i+1} + \mathcal{O}(\Delta x^2) + \mathcal{O}(\Delta t^2) \\ &= a_{i+1}^n - \frac{\Delta x}{2}\left.\frac{\partial a}{\partial x}\right|_{i+1} + \frac{\Delta t}{2}\left(-u\left.\frac{\partial a}{\partial x}\right|_{i+1}\right) + \ldots \\ &= a_{i+1}^n - \frac{\Delta x}{2}\left(1 + \frac{\Delta t}{\Delta x}u\right)\left.\frac{\partial a}{\partial x}\right|_{i+1} + \ldots \end{aligned} \tag{3.19}$$

A suitable estimate is needed for the slope of a that appears in these expressions (as $\partial a/\partial x$). We can approximate this simply as

$$\left.\frac{\partial a}{\partial x}\right|_i = \frac{a_{i+1} - a_{i-1}}{2\Delta x} \tag{3.20}$$

We can think of this method as reconstructing the function form of the data from the cell-average data in each cell using a piecewise linear polynomial.

We now have two states, $a^{n+1/2}_{i+1/2,L}$ and $a^{n+1/2}_{i+1/2,R}$ separated by an interface—this is called the *Riemann problem*. The solution to this will depend on the equation being solved, and results in a single state at the interface:

$$a^{n+1/2}_{i+1/2} = \mathcal{R}(a^{n+1/2}_{i+1/2,L}, a^{n+1/2}_{i+1/2,R}) \tag{3.21}$$

In our case, the advection equation simply propagates the state to the right (for $u > 0$), so the solution to the Riemann problem is to take the left state (this is another example of upwinding). That is we do:

$$\mathcal{R}(a^{n+1/2}_{i+1/2,L}, a^{n+1/2}_{i+1/2,R}) = \begin{cases} a^{n+1/2}_{i+1/2,L} & u > 0 \\ a^{n+1/2}_{i+1/2,R} & u < 0 \end{cases} \tag{3.22}$$

To complete the update, we use this interface state to evaluate the flux and update the advected quantity via Eq. 3.16.

Boundary conditions are implemented by filling the ghost cells outside each end of the domain based on data in the interior. Note that at the very left edge of the domain, the state $a^{n+1/2}_{\mathrm{lo}-1/2}$ requires the construction of states on the left and right. The left state at that interface, $a^{n+1/2}_{\mathrm{lo}-1/2,L}$ depends on the slope reconstructed in the $\mathrm{lo}-1$ ghost cell, $\partial a/\partial x|_{\mathrm{lo}-1}$. This in turn is constructed using a limited center-difference that will consider the value in the cell to the left, $\mathrm{lo}-2$. Therefore, we need two ghost cells at each end of the domain for this method—figure 3.6 illustrates this. Higher-order limiters may require even more ghost cells.

Exercise 3.6: *Write a second-order solver for the linear advection equation. To mimic a real hydrodynamics code, your code should have routines for finding initializing the state, filling boundary conditions, computing the timestep, computing the interface states, solving the Riemann problem, and doing the update. The problem flow should look like:*

- *set initial conditions*
- *main evolution loop—loop until final time reached*
  - *fill boundary conditions*
  - *get timestep (Eq. 3.5)*

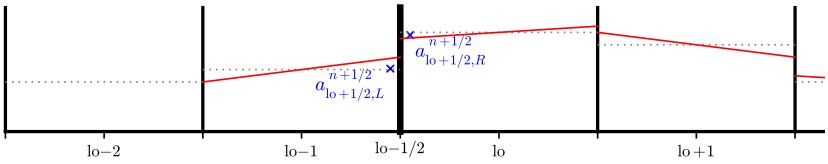

Figure 3.6: Reconstruction near the boundary, showing the need for two ghostcells. Here we see the left and right state at the left physical boundary of the domain (marked as lo $-$ 1/2). The gray dotted lines are the piecewise constant cell averages and the red lines are the reconstructed slopes. Note that we need the slope in lo $-$ 1 to get the left interface state at lo $-$ 1/2, and that slope in turn needed the data in zone lo $-$ 2 to construct a centered-difference.

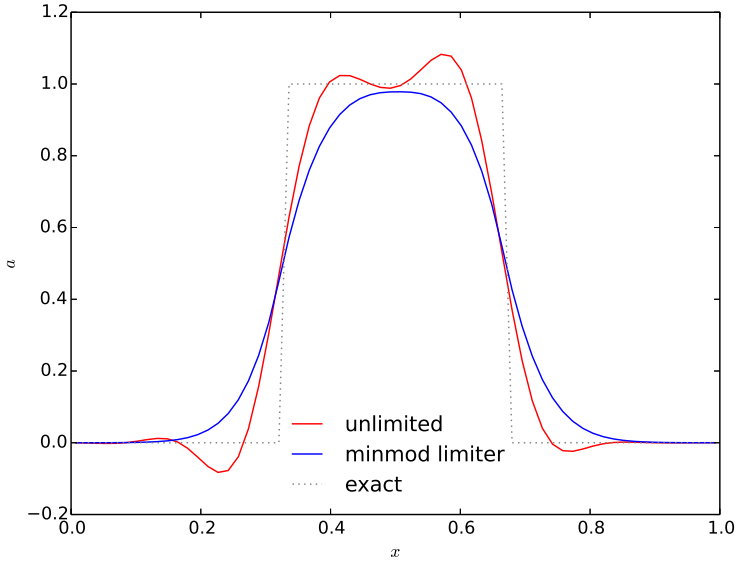

Figure 3.7: Second-order finite volume advection showing the result of advecting a tophat profile through five periods with both unlimited and limited slopes. This calculation used 64 zones and $C = 0.7$.
hydro_examples: `advection.py`

- *compute interface states (Eqs. 3.18 and 3.19)*
- *solve Riemann problem at all interfaces (Eq. 3.22)*
- *do conservative update (Eq. 3.16)*

Use both the top-hat and Gaussian initial conditions and periodic boundary conditions and compare to the first-order method. See Figure 3.7.

3.3.1 Limiting

The second-order method likely showed some oscillations in the solution, especially for the top-hat initial conditions. *Godunov's theorem* says that any monotonic linear method for advection is first-order accurate (see, e.g, [31]). In this context, monotonic means that no new minima or maxima are introduced. The converse is true too, which suggests that in order to have a second-order accurate method for this linear equation, the algorithm itself must be nonlinear.

Exercise 3.7: *To remove the oscillations in practice, we limit the slopes to ensure that no new minima or maxima are introduced during the advection process. There are many choices for limited slopes. A popular one is the* minmod *limiter. Here, we construct the slopes in the interface states as:*

$$\left.\frac{\partial a}{\partial x}\right|_i = \texttt{minmod}\left(\frac{a_i - a_{i-1}}{\Delta x}, \frac{a_{i+1} - a_i}{\Delta x}\right) \tag{3.23}$$

instead of Eq. 3.20. with

$$\texttt{minmod}(a,b) = \begin{cases} a & \text{if } |a| < |b| \text{ and } a \cdot b > 0 \\ b & \text{if } |b| < |a| \text{ and } a \cdot b > 0 \\ 0 & \text{otherwise} \end{cases} \tag{3.24}$$

Use this slope in your second-order advection code and notice that the oscillations go away—see Figure 3.7.

We can get a feel for what happens with and without limiting pictorially. Figures 3.8 and 3.9 show the evolution of an initial discontinuity with and without limiting. See the text by LeVeque [32] for alternate choices of limiters. Note: most limiters will have some sort of test on the product of a left-sided and right-sided difference ($a \cdot b$ above)—this is < 0 at an extremum, which is precisely where we want to limit.

A slightly more complex limiter is the MC limiter (monotonized central difference). First we define an extrema test,

$$\xi = (a_{i+1} - a_i) \cdot (a_i - a_{i-1}) \tag{3.25}$$

Then the limited difference is

$$\left.\frac{\partial a}{\partial x}\right|_i = \begin{cases} \min\left[\frac{|a_{i+1}-a_{i-1}|}{2\Delta x}, 2\frac{|a_{i+1}-a_i|}{\Delta x}, 2\frac{|a_i-a_{i-1}|}{\Delta x}\right] \text{sign}(a_{i+1} - a_{i-1}) & \xi > 0 \\ 0 & \text{otherwise} \end{cases} \tag{3.26}$$

Note that a slightly different form of this limiter is presented in [32], where all quantities are in a `minmod`, which appears to limit a bit less. This is second-order accurate for smooth flows.

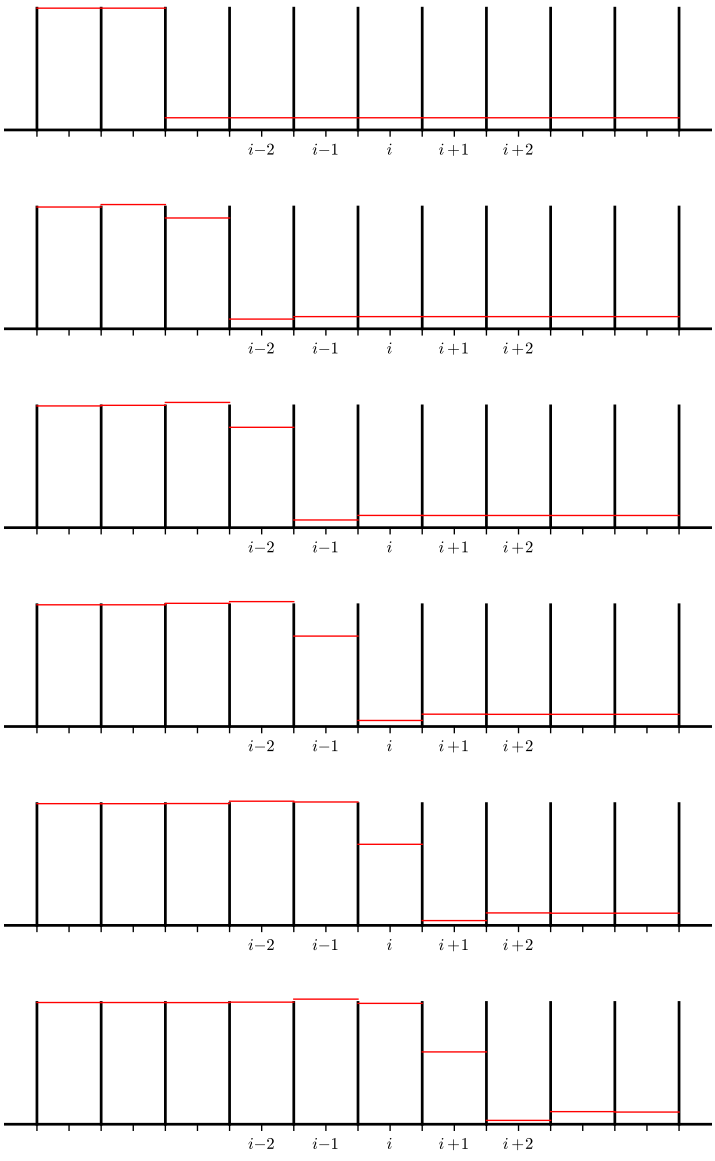

Figure 3.8: Initially discontinuous data evolved for several steps with no limiting. Notice that there are overshoots and undershoots surrounding the discontinuity.

The main goal of a limiter is to reduce the slope near extrema. Figure 3.10 shows a finite-volume grid with the original data, cell-centered slopes, and MC limited slopes. Note that near the strong gradients is where the limiting kicks in. The different limiters are all constructed by enforcing a condition requiring the method to be *total variation diminishing*, or TVD. More details on TVD limiters can be found in [51, 32].

A popular extension of the MC limiter is the 4^{th}-order MC limiter, which is more accurate in smooth flows (this is shown in [18], Eqs. 2.5 and 2.6; and [19], Eq. 191).

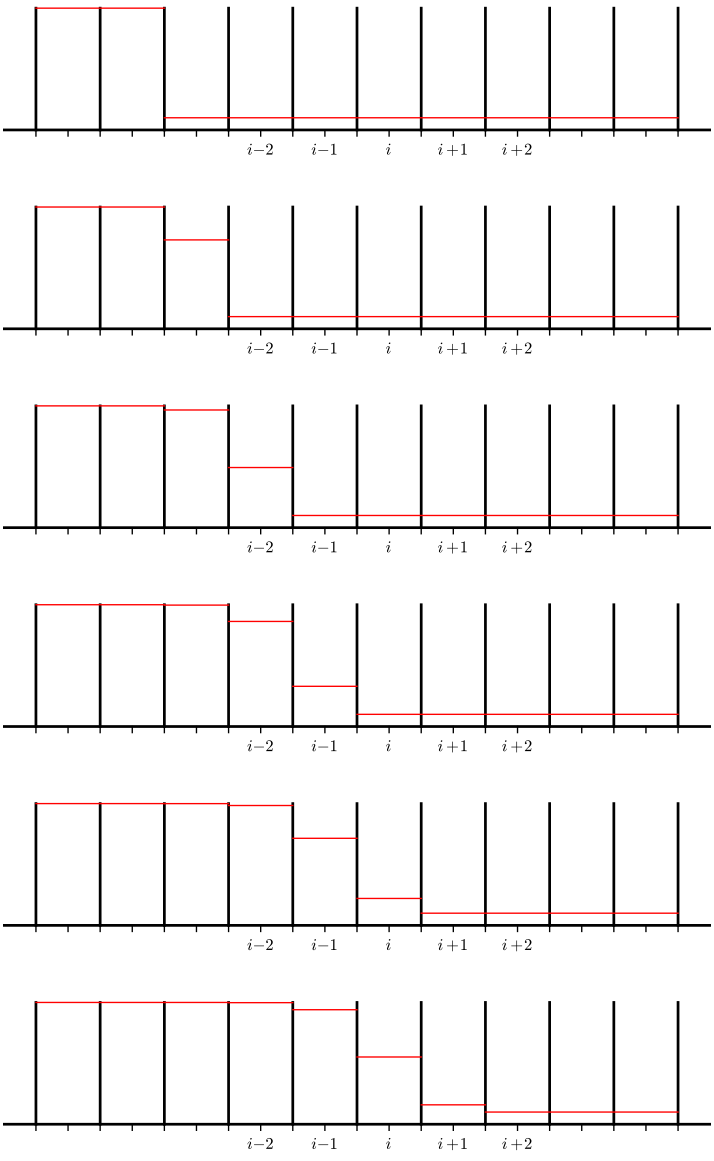

Figure 3.9: Initially discontinuous data evolved for several steps with limiting. Note that unlike the sequence without limiting (Figure 3.8), the discontinuity remains sharper with limiting and there are no over- or undershoots.

Exercise 3.8: *Show analytically that if you fully limit the slopes (i.e. set* $\partial a/\partial x|_i = 0$*, that the second-order method reduces to precisely our first-order finite-difference discretization, Eq. 3.2.*

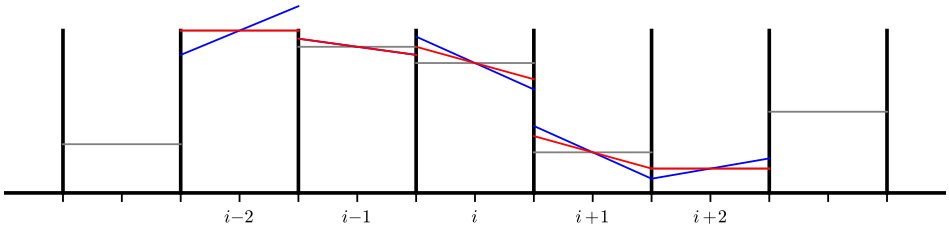

Figure 3.10: A finite-volume grid showing the cell averages (gray horizontal lines), unlimited center-difference slopes (blue) and MC limited slopes (red). Note that in zones i and $i+1$, the slopes are limited slightly, so as not to overshoot or undershoot the neighboring cell value. Cell $i-1$ is not limited at all, whereas cells $i-2$, and $i+2$ are fully limited—the slope is set to 0—these are extrema.

3.3.2 Reconstruct-evolve-average

Another way to think about these methods is as a reconstruction, evolve, and average (R-E-A) process (see Figure 3.11).

We can write the conservative update as:

$$a_i^{n+1} = a_i^n + \frac{\Delta t}{\Delta x}(ua_{i-1/2}^{n+1/2} - ua_{i+1/2}^{n+1/2}) \tag{3.27}$$

$$= a_i^n + C(a_{i-1/2}^{n+1/2} - a_{i+1/2}^{n+1/2}) \tag{3.28}$$

If we take $u > 0$, then the Riemann problem will always choose the left state, so we can write this as:

$$a_i^{n+1} = a_i^n + C\left[\underbrace{\left(a_{i-1}^n + \frac{1}{2}(1-C)\Delta a_{i-1}\right)}_{a_{i-1/2,L}} - \underbrace{\left(a_i^n + \frac{1}{2}(1-C)\Delta a_i\right)}_{a_{i+1/2,L}}\right] \tag{3.29}$$

If we instead look at this via the R-E-A procedure, we write the reconstructed a in each zone in the form of a piecewise linear polynomial

$$a_i(x) = a_i + \frac{\Delta a_i}{\Delta x}(x - x_i) \tag{3.30}$$

Consider zone i. If we are advecting with a CFL number C, then that means that the fraction C of the zone immediately to the left of the $i-1/2$ interface will advect into zone i over the timestep. And only the fraction $1-C$ in zone i immediately to the right of the interface will stay in that zone. This is indicated by the shaded regions in Figure 3.11.

The average of the quantity a from zone $i-1$ that will advect into zone i is

$$\begin{aligned}
\mathcal{I}_< &= \frac{1}{C\Delta x}\int_{x_{i-1/2}-C\Delta x}^{x_{i-1/2}} a_{i-1}(x)dx && (3.31)\\
&= \frac{1}{C\Delta x}\int_{x_{i-1/2}-C\Delta x}^{x_{i-1/2}} \left[a_{i-1} + \frac{\Delta a_{i-1}}{\Delta x}(x - x_{i-1})\right] dx && (3.32)\\
&= a_{i-1} + \frac{1}{2}\Delta a_{i-1}(1-C) && (3.33)
\end{aligned}$$

And the average of the quantity a in zone i that will remain in zone i is

$$\begin{aligned}
\mathcal{I}_> &= \frac{1}{(1-C)\Delta x}\int_{x_{i-1/2}}^{x_{i-1/2}+(1-C)\Delta x} a_i(x)dx && (3.34)\\
&= \frac{1}{(1-C)\Delta x}\int_{x_{i-1/2}}^{x_{i-1/2}+(1-C)\Delta x} \left[a_i + \frac{\Delta a_i}{\Delta x}(x - x_i)\right] dx && (3.35)\\
&= a_i - \frac{1}{2}\Delta a_i C && (3.36)
\end{aligned}$$

The final part of the R-E-A procedure is to average the over the advected profiles in the new cell. The weighted average of the amount brought in from the left of the interface and that that remains in the cell is

$$a_i^{n+1} = C\mathcal{I}_< + (1-C)\mathcal{I}_> \tag{3.37}$$

This is identical to Eq. 3.29. This demonstrates that the R-E-A procedure is equivalent to our reconstruction, prediction of the interface states, solving the Riemann problem, and doing the conservative flux update.

3.4 Errors and convergence rate

For the advection problem (with $u > 0$), the analytic solution is to simply propagate the initial profile to the right. This means that with periodic boundary conditions, after advecting for one period, our numerical solution should be identical to the initial conditions. Any differences are our numerical error. We can quantify the error by taking the norm of error as:

$$\epsilon^{\text{abs}} = \|a^{\text{final}} - a^{\text{init}}\|_2 \equiv \left[\frac{1}{N}\sum_{i=1}^{N}(a_i^{\text{final}} - a_i^{\text{init}})^2\right]^{1/2} \tag{3.38}$$

It is sometimes useful to compare to the norm of the original solution to get a measure of the relative error:

$$\epsilon^{\text{rel}} \equiv \frac{\|a^{\text{final}} - a^{\text{init}}\|_2}{\|a^{\text{init}}\|_2} \tag{3.39}$$

Note that it is important in these definitions to normalize by the number of zones, N, otherwise our error will be resolution-dependent.

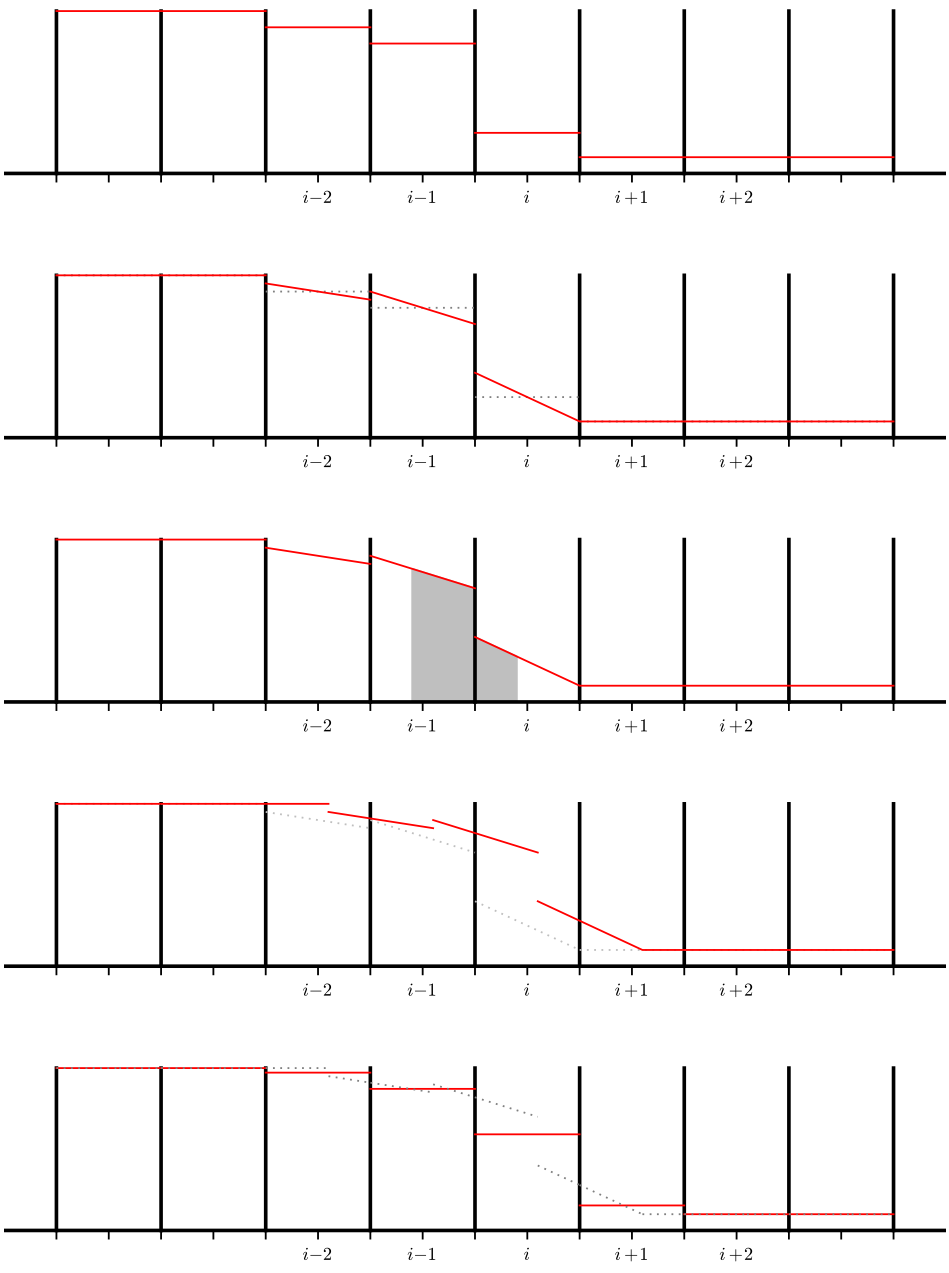

Figure 3.11: Reconstruct-Evolve-Average. The top panel shows the original cell-average data. The second panel shows the (limited) piecewise linear reconstruction of the data. Assuming a CFL number of 0.6 and advection to the right, the shaded regions in the third panel show the data that will wind up in cell i after advecting for a single step. The fourth panel shows the piecewise-linear data advected to the right by 0.6 of a cell-width (corresponding to a CFL of 0.6). The final panel shows the new averages of the data, constructed by averaging the advected piecewise linear data in each cell.

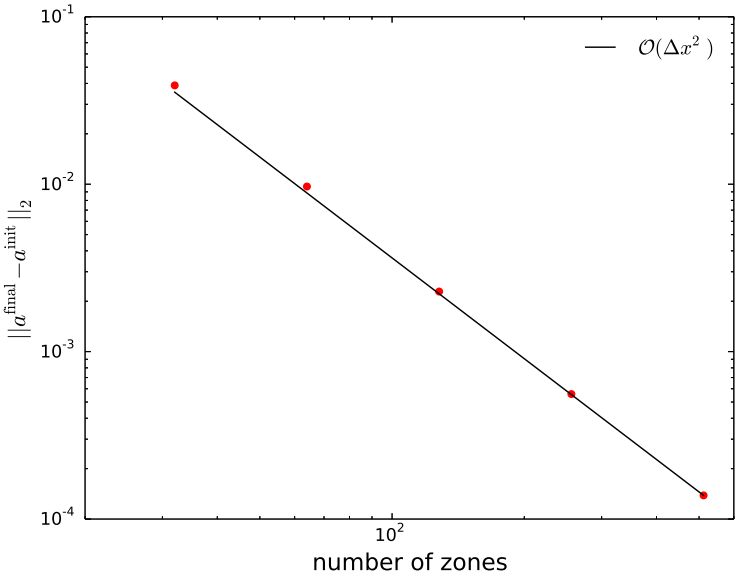

Figure 3.12: Convergence for the second-order finite-volume method with minmod limiting advecting a Gaussian initial profile with $C = 0.8$.
hydro_examples: `advection.py`

Exercise 3.9: *Run the first-order solver for several different Δxs, each a factor of 2 smaller than the previous. Compute ϵ for each resolution and observe that it converges in a first-order fashion (i.e. ϵ decreases by 2 when we decrease Δx by a factor of 2).*

Do the same with the second-order solver and observe that it converges as second-order. However: *you may find less than second-order is your initial conditions have discontinuities and you are limiting. Figure 3.12 shows the convergence of the method with minmod limiting, $C = 0.8$, and a Gaussian initial condition.*

3.5 Multi-dimensional advection

The two-dimensional linear advection equation is:

$$a_t + ua_x + va_y = 0 \tag{3.40}$$

where u is the velocity in the x-direction and v is the velocity in the y-direction. We denote the average of $a(x, y, t)$ in a zone i, j as $a_{i,j}$. Here, i is the index in the x-direction and j is the index in the y-direction. A 2-d grid is shown in Figure 3.13. Just as in the one-dimensional case, we will extend the domain with a perimeter of ghost cells to set the boundary conditions.

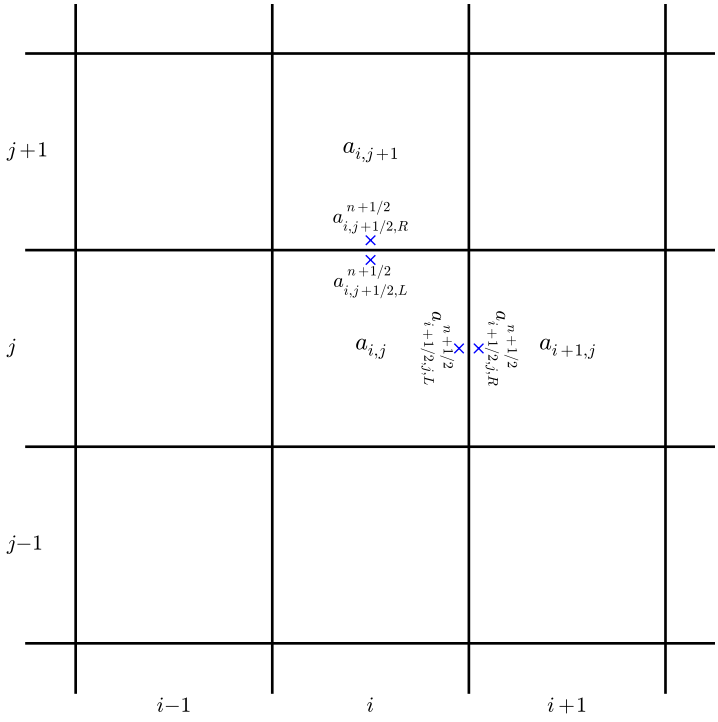

Figure 3.13: A simple 2-d grid with the zone-centered indexes. The $\times$s mark the interface states at the upper edge of the i, j zone in each coordinate direction.

To derive the finite-volume form of the update, we start by writing this in conservative form. Since u and v are constant, we can move them inside the divergences:

$$a_t + (ua)_x + (va)_y = 0 \tag{3.41}$$

This is the form we will integrate over zones. As before, we will define the average of a in a zone by integrating it over the volume:

$$a_{i,j} = \frac{1}{\Delta x \Delta y} \int_{x_{i-1/2}}^{x_{i+1/2}} \int_{y_{j-1/2}}^{y_{j+1/2}} a(x, y, t)\, dx\, dy \tag{3.42}$$

Integrating Eq. 3.41 over x and y, we have:

$$\begin{aligned} \frac{1}{\Delta x \Delta y} \int_{x_{i-1/2}}^{x_{i+1/2}} \int_{y_{j-1/2}}^{y_{j+1/2}} a_t\, dx\, dy = &- \frac{1}{\Delta x \Delta y} \int_{x_{i-1/2}}^{x_{i+1/2}} \int_{y_{j-1/2}}^{y_{j+1/2}} (ua)_x\, dx\, dy \\ &- \frac{1}{\Delta x \Delta y} \int_{x_{i-1/2}}^{x_{i+1/2}} \int_{y_{j-1/2}}^{y_{j+1/2}} (va)_y\, dx\, dy \end{aligned} \tag{3.43}$$

or using the divergence theorem,

$$\frac{\partial a_{i,j}}{\partial t} = -\frac{1}{\Delta x \Delta y}\int_{y_{j-1/2}}^{y_{j+1/2}} \left\{(ua)_{i+1/2,j} - (ua)_{i-1/2,j}\right\} dy - \frac{1}{\Delta x \Delta y}\int_{x_{i-1/2}}^{x_{i+1/2}} \left\{(va)_{i,j+1/2} - (va)_{i,j-1/2}\right\} dx \tag{3.44}$$

Now we integrate over time—the left side of our expression becomes just the different between the new and old state.

$$a_{i,j}^{n+1} - a_{i,j}^{n} = -\frac{1}{\Delta x \Delta y}\int_{t^n}^{t^{n+1}}\int_{y_{j-1/2}}^{y_{j+1/2}} \left\{(ua)_{i+1/2,j} - (ua)_{i-1/2,j}\right\} dydt - \frac{1}{\Delta x \Delta y}\int_{t^n}^{t^{n+1}}\int_{x_{i-1/2}}^{x_{i+1/2}} \left\{(va)_{i,j+1/2} - (va)_{i,j-1/2}\right\} dxdt \tag{3.45}$$

We define the flux through the interface as the average over the face of that interface and time:

$$(ua)_{i+1/2,j} = \frac{1}{\Delta y \Delta t}\int_{t^n}^{t^{n+1}}\int_{y_{j-1/2}}^{y_{j+1/2}} (ua)_{i+1/2,j} dydt \tag{3.46}$$

$$(va)_{i,j+1/2} = \frac{1}{\Delta x \Delta t}\int_{t^n}^{t^{n+1}}\int_{x_{i-1/2}}^{x_{i+1/2}} (va)_{i,j+1/2} dxdt \tag{3.47}$$

For a second-order accurate method in time, we replace the integration in time with the flux at the midpoint in time, and in space, we replace the integration with the center of the interface. Then we have:

$$a_{i,j}^{n+1} = a_{i,j}^{n} - \frac{(ua)_{i+1/2,j}^{n+1/2} - (ua)_{i-1/2,j}^{n+1/2}}{\Delta x} - \frac{(va)_{i,j+1/2}^{n+1/2} - (va)_{i,j-1/2}^{n+1/2}}{\Delta y} \tag{3.48}$$

For the advection problem, since u and v are constant, we need to find the interface states of a alone. There are two methods for computing these interface states, $a_{i\pm1/2,j}^{n+1/2}$ on x-interfaces and $a_{i,j\pm1/2}^{n+1/2}$ on y-interfaces: dimensionally split and unsplit. Dimensionally split methods are easier to code, since each dimension is operated on independent of the others, so you can simply call a one-dimensional method for each direction. Unsplit methods, however, are more accurate and less susceptible to grid effects.

3.5.1 Dimensionally split

In a split method, we update the state in each coordinate direction independently. This is simple and a straightforward way to use one-dimensional methods in multi-d. To be second-order accurate in time, we do *Strang splitting* [49], where we

alternate the order of the dimensional updates each timestep. An update through Δt consists of x and y sweeps and appears as:

$$\frac{a^\star_{i,j} - a^n_{i,j}}{\Delta t} = -\frac{ua^{n+1/2}_{i+1/2,j} - ua^{n+1/2}_{i-1/2,j}}{\Delta x} \tag{3.49}$$

$$\frac{a^{n+1}_{i,j} - a^\star_{i,j}}{\Delta t} = -\frac{va^{\star,n+1/2}_{i,j+1/2} - va^{\star,n+1/2}_{i,j-1/2}}{\Delta y} \tag{3.50}$$

Here, Eq. 3.49 is the update in the x-direction. In constructing the interface states, $a^{n+1/2}_{i+1/2,j}$ and $a^{n+1/2}_{i-1/2,j}$, we do the exact same procedure as the one-dimensional case, constructing the left and right states at each interface and then solving the same Riemann problem to find the unique state on the interface. Each dimensional sweep is done without knowledge of the other dimensions. For example, in the x-update, we are solving:

$$a_t + ua_x = 0 \tag{3.51}$$

and in the y-update, we are solving:

$$a_t + va_y = 0 \tag{3.52}$$

The construction of the interface states largely mimics the one-dimensional case (Eq. 3.18 and 3.19). For example, the $a^{n+1/2}_{i+1/2,j,L}$ state is:

$$\begin{aligned} a^{n+1/2}_{i+1/2,j,L} &= a^n_{i,j} + \frac{\Delta x}{2}\left.\frac{\partial a}{\partial x}\right|_{i,j} + \frac{\Delta t}{2}\left.\frac{\partial a}{\partial t}\right|_{i,j} + \mathcal{O}(\Delta x^2) + \mathcal{O}(\Delta t^2) \\ &= a^n_{i,j} + \frac{\Delta x}{2}\left.\frac{\partial a}{\partial x}\right|_{i,j} + \frac{\Delta t}{2}\left(-u\left.\frac{\partial a}{\partial x}\right|_{i,j}\right) + \dots \\ &= a^n_{i,j} + \frac{\Delta x}{2}\left(1 - \frac{\Delta t}{\Delta x}u\right)\left.\frac{\partial a}{\partial x}\right|_{i,j} + \dots \end{aligned} \tag{3.53}$$

Notice that when substituting for $\partial a/\partial t$, we use the one-dimensional split version of the advection equation (Eq. 3.51) instead of the full multi-dimensional equation. There are no y-direction terms that come into play in the split method when considering the x-direction.

The x-update (Eq. 3.49) updates the state only accounting for the x-fluxes—we denote this intermediate state with the '$\star$' superscript. For the y-update, we construct our interface states in the analogous way as in the x-direction, but begin with the '$\star$' state instead of the old-time state. In this fashion, the y-update 'sees' the result of the x-update and couples things together.

To achieve second-order accuracy in time, it is necessary to alternate the directions of the sweeps each timestep, i.e. x-y then y-x. Furthermore, this pair of sweeps should use the same timestep, Δt.

3.5.2 Unsplit multi-dimensional advection

The unsplit case differs from the dimensionally split case in two ways: (1) in predicting the interface states, we use knowledge of the flow in the transverse direction, and (2), only a single conservative update is done per timestep, with all directions updating simultaneously. See [19] for more details. This idea is sometimes called the "corner transport upwind" or CTU method.

The construction of the $a^{n+1/2}_{i+1/2,j,L}$ interface state appears as

$$\begin{aligned} a^{n+1/2}_{i+1/2,j,L} &= a^n_{i,j} + \frac{\Delta x}{2}\left.\frac{\partial a}{\partial x}\right|_{i,j} + \frac{\Delta t}{2}\left.\frac{\partial a}{\partial t}\right|_{i,j} + \mathcal{O}(\Delta x^2) + \mathcal{O}(\Delta t^2) \\ &= a^n_{i,j} + \frac{\Delta x}{2}\left.\frac{\partial a}{\partial x}\right|_{i,j} + \frac{\Delta t}{2}\left(-u\left.\frac{\partial a}{\partial x}\right|_{i,j} - v\left.\frac{\partial a}{\partial y}\right|_{i,j}\right) + \dots \\ &= a^n_{i,j} + \frac{\Delta x}{2}\left(1 - \frac{\Delta t}{\Delta x}u\right)\left.\frac{\partial a}{\partial x}\right|_{i,j} \underbrace{-\frac{\Delta t}{2}v\left.\frac{\partial a}{\partial y}\right|_{i,j}}_{\text{"transverse flux difference"}} + \dots \end{aligned} \tag{3.54}$$

The main difference between the split and unsplit interface states is the explicitly appearance of the "transverse flux difference" in the unsplit interface state. We rewrite this as:

$$a^{n+1/2}_{i+1/2,j,L} = \hat{a}^{n+1/2}_{i+1/2,j,L} - \frac{\Delta t}{2}v\left.\frac{\partial a}{\partial y}\right|_{i,j} \tag{3.55}$$

Here, the $\hat{a}^{n+1/2}_{i+1/2,j,L}$ term is just the normal prediction without considering the transverse direction. The basic update procedure is:

- Construct the normal predictor term, $\hat{a}^{n+1/2}_{i+1/2,j,L}$, in a fashion identical to the one-dimensional and split method. We compute these one-dimensional $\hat{a}$'s at the left and right every interface in both coordinate directions. Note that these states are still one-dimensional, since we have not used any information from the transverse direction in their computation.

- Solve a Riemann problem at each of these interfaces:

$$a^T_{i+1/2,j} = \mathcal{R}(\hat{a}^{n+1/2}_{i+1/2,j,L}, \hat{a}^{n+1/2}_{i+1/2,j,R}) \tag{3.56}$$

$$a^T_{i,j+1/2} = \mathcal{R}(\hat{a}^{n+1/2}_{i,j+1/2,L}, \hat{a}^{n+1/2}_{i,j+1/2,R}) \tag{3.57}$$

$$\tag{3.58}$$

  These states are given the '^{T}' superscript since they are the states that are used in computing the transverse flux difference.

- Correct the previously computed normal interface states (the $\hat{a}$'s) with the transverse flux difference:

$$a^{n+1/2}_{i+1/2,j,L} = \hat{a}^{n+1/2}_{i+1/2,j,L} - \frac{\Delta t}{2}v\frac{a^T_{i,j+1/2} - a^T_{i,j-1/2}}{\Delta y} \tag{3.59}$$

A similar procedure happens at the y-interfaces. Notice that the fluxes that are differenced for the left state are those that are transverse, but to the left of the interface. Similarly, for the right state, it would be those that are transverse, but to the right of the interface:

$$a^{n+1/2}_{i+1/2,j,R} = \hat{a}^{n+1/2}_{i+1/2,j,R} - \frac{\Delta t}{2} v \frac{a^T_{i+1,j+1/2} - a^T_{i+1,j-1/2}}{\Delta y} \tag{3.60}$$

Figure 3.14 illustrates the steps involved in the construction of the $a^{n+1/2}_{i+1/2,j,L}$ state.

Once all of the full states (normal prediction + transverse flux difference) are computed to the left and right of all the interfaces (x and y), we solve another Riemann problem to find the final state on each interface.

$$a^{n+1/2}_{i+1/2,j} = \mathcal{R}(a^{n+1/2}_{i+1/2,j,L}, a^{n+1/2}_{i+1/2,j,R}) \tag{3.61}$$

The final conservative update is then done via Eq. 3.45.

See [19] for more details on this unsplit method.

3.6 Going further

- *Stability analysis*: many texts provide the details of von Neumann stability analysis for the linear advection equation. There, you will see how a Fourier analysis can be used to derive the CFL condition. Note: this stability analysis is restricted to linear equations, but we nevertheless use the resulting limits for nonlinear equations (like the Euler equations).
- *Slope limiting*: there are a wide variety of slope limiters. All of them are designed to reduce oscillations in the presence of discontinuities or extrema, but some are higher-order and can be less restrictive when dealing with smooth flows. Most hydro texts (e.g. [32, 51]) provide an introduction to the design of such limiters.
- *Multi-dimensional limiting*: the procedure described above still does the limiting in each dimension independent of the other when doing the unsplit reconstruction. This can lead to overshoots/ undershoots. An example of a method that considers the limiting in multi-dimensions is [10, 35].
- *Spatially-varying velocity field*: if we consider a spatially varying velocity field, $u(x, y)$ and $v(x, y)$ that is specified externally, then we can describe the advection of a quantity ϕ as:

$$\phi_t + (\phi u)_x + (\phi v)_y = 0 \tag{3.62}$$

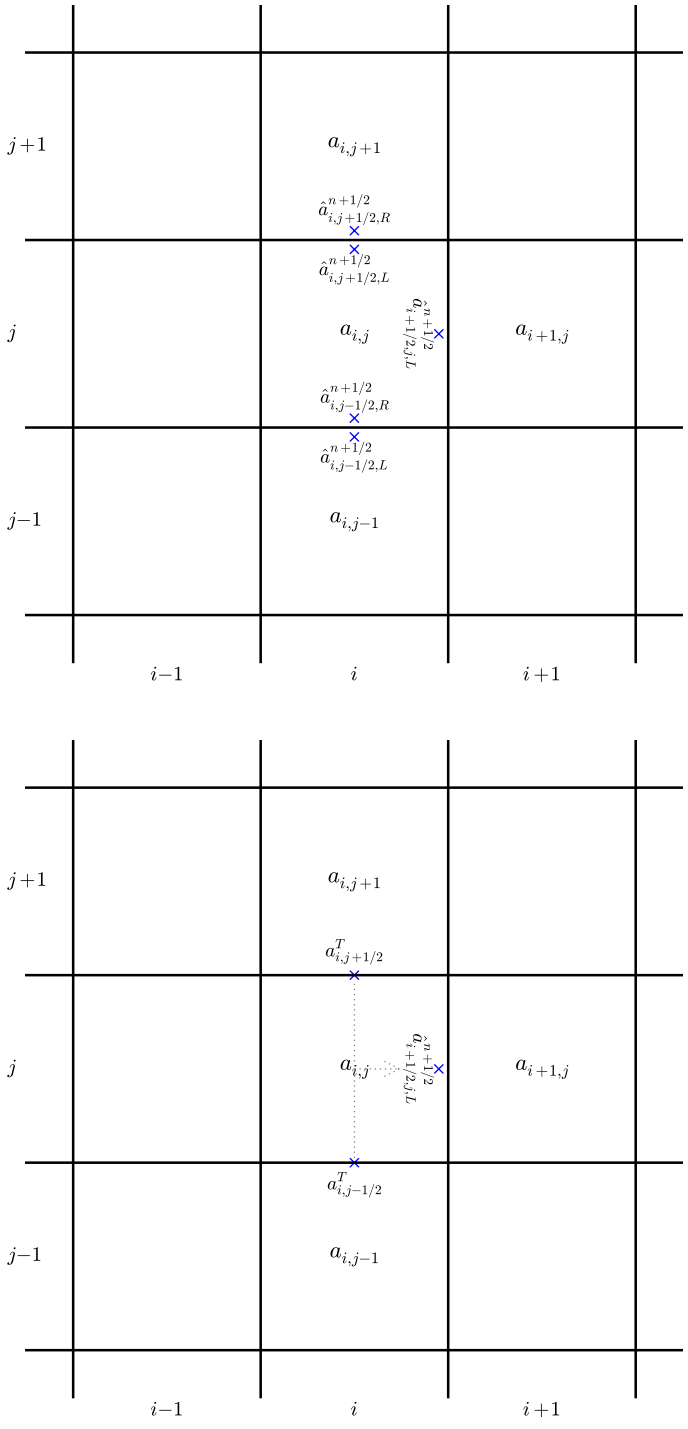

Figure 3.14: The construction of the $a^{n+1/2}_{i+1/2,j,L}$ state. Top: first we compute the $\hat{a}$'s—here we show all of the $\hat{a}$'s that will be used in computing the full left interface state at $(i+1/2, j)$. Bottom: after the transverse Riemann solves, we have the two transverse states ($a^T_{i,j+1/2}$ and $a^T_{i,j-1/2}$) that will be differenced and used to correct $\hat{a}^{n+1/2}_{i+1/2,j,L}$ (illustrated by the dotted lines) to make $a^{n+1/2}_{i+1/2,j,L}$.

The solution procedure is largely the same as described above. We write:

$$\begin{aligned}\phi^{n+1/2}_{i+1/2,j,L} &= \phi^n_{i,j} + \frac{\Delta x}{2}\frac{\partial \phi}{\partial x} + \frac{\Delta t}{2}\frac{\partial \phi}{\partial t} + \ldots \\ &= \phi^n_{i,j} + \frac{\Delta x}{2}\frac{\partial \phi}{\partial x} + \frac{\Delta t}{2}\left[-(\phi u)_x - (\phi v)_y\right]_{i,j} \\ &= \underbrace{\phi^n_{i,j} + \frac{\Delta x}{2}\left(1 - \frac{\Delta t}{\Delta x}u_{i,j}\right)\frac{\partial \phi}{\partial x}}_{\hat{\phi}^{n+1/2}_{i+1/2,j,L}} - \frac{\Delta t}{2}\left[\phi u_x\right]_{i,j} - \frac{\Delta t}{2}\left[(\phi v)_y\right]_{i,j} \end{aligned} \quad (3.63)$$

and upwinding is used to resolve the Riemann problem for both the transverse and normal interface states.

For compressible hydrodynamics, we often have density-weighted quantities that we advect. This extension is described in § 5.9. For low Mach number flows, the density can be advected according to the velocity field in much the fashion shown here, as described in § 3.

3.7 pyro experimentation

To gain some experiences with these ideas, we can use the advection solver in pyro (see Appendix B to get started). The pyro advection solver implements the second-order unsplit advection algorithm described in the previous sections. To run this solver on the Gaussian advection problem, do:

```
./pyro.py advection smooth inputs.smooth
```

By default, this will advect a Gaussian profile diagonally across the domain for a single period.

To get a feel for the advection algorithm, here are some suggested exercises:

- Implement a tophat initial profile and run with and without limiters (this is controlled by the `advection.limiter` runtime parameter.
- Look at the solution when you advect purely in the x- or y-direction and compare to the diagonal case—notice how the direction affects the error in the solution.
- Implement a dimensionally-split version of the advection algorithm and compare the results to the unsplit version.

Chapter 4

Burgers' Equation

These notes extend our ideas of linear advection to a scalar nonlinear equation.

4.1 Burgers' equation

The inviscid Burgers' equation is the simplest *nonlinear* hyperbolic equation:

$$u_t + uu_x = 0 \tag{4.1}$$

Here u is both the quantity being advected and the speed at which it is moving. In conservative form, this appears as:

$$u_t + \left[\tfrac{1}{2}u^2\right]_x = 0 \tag{4.2}$$

The solution of this follows the same methodology as outlined above. The interface states are predicted as:

$$\begin{aligned} u^{n+1}_{i+1/2,L} &= u^n_i + \frac{\Delta x}{2}\frac{\partial u}{\partial x} + \frac{\Delta t}{2}\left.\frac{\partial u}{\partial t}\right|_i + \ldots && (4.3)\\ &= u^n_i + \frac{\Delta x}{2}\frac{\partial u}{\partial x} + \frac{\Delta t}{2}\left.\left(-u_i\frac{\partial u}{\partial x}\right)\right|_i + \ldots && (4.4)\\ &= u^n_i + \frac{\Delta x}{2}\left(1 - \frac{\Delta t}{\Delta x}u_i\right)\left.\frac{\partial u}{\partial x}\right|_i + \ldots && (4.5) \end{aligned}$$

The only difference with the linear advection equation is that now $u_i\Delta t/\Delta x$ varies from zone to zone, whereas with linear advection, it is the constant C. The slopes are computed using the same limiters as with linear advection.

add characteris tracing

The Riemann problem differs from linear advection. It remains the case that the solution is constant along the lines $x = ut + x_0$ (the characteristics), but now those

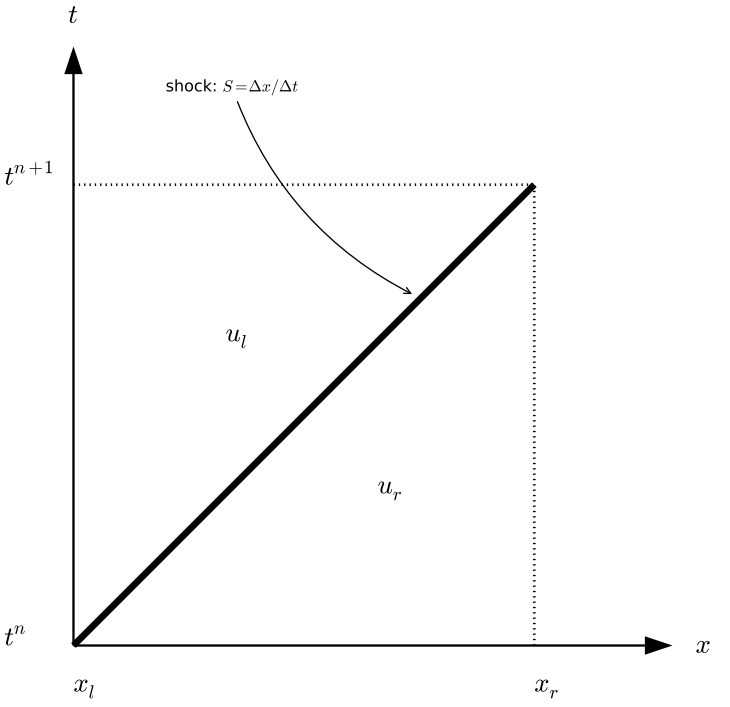

Figure 4.1: A rightward moving shock in the x-t plane separating two states: u_l and u_r.

lines are no longer parallel. If the characteristic lines intersect, then there it is not possible to trace backward from time to learn where the flow originated. This is the condition for a *shock*.

The shock speed is computed through the *Rankine-Hugoniot* jump conditions. For a scalar equation, these are easy to construct. We'll follow the method of [32]. Figure 4.1 shows two states separated by a rightward moving shock in the x-t plane. At time t^n, the state in our interval ($x \in [x_l, x_r]$) is entirely u_r, and at the later time, t^{n+1} it is entirely u_l. The shock moves with a speed $S = \Delta x/\Delta t$ in this figure. To determine the speed, we integrate our conservation law over both space and time (and normalize by $\Delta x = x_r - x_l$):

$$\frac{1}{\Delta x}\int_{x_l}^{x_r} dx \int_{t^n}^{t^{n+1}} dt u_t = -\frac{1}{\Delta x}\int_{x_l}^{x_r} dx \int_{t^n}^{t^{n+1}} dt \left[f(u)\right]_x \tag{4.6}$$

Doing the t integral on the left and x integral on the right, we have

$$\frac{1}{\Delta x}\int_{x_l}^{x_r} \left\{u(t^{n+1}) - u(t^n)\right\} dx = -\frac{1}{\Delta x}\int_{t^n}^{t^{n+1}} \left\{f(u)|_{x=x_r} - f(u)|_{x=x_l}\right\} dt \tag{4.7}$$

Recognizing that at $t = t^n$, $u = u_r$ and at $t = t^{n+1}$, $u = u_l$, $\{u(t^{n+1}) - u(t^n)\}$ in the left side becomes $\{u_l - u_r\}$. For the right side, we see that all along $x = x_l$ the flux

is $f = f(u_l)$ for $t \in [t^n, t^{n+1}]$. Likewise, all along $x = x_r$, the flux is $f = f(u_r)$ in the same time interval (see the figure). Therefore, our expression becomes:

$$(u_l - u_r) = -\frac{\Delta t}{\Delta x}\left[f(u_r) - f(u_l)\right] \tag{4.8}$$

and using $S = \Delta x/\Delta t$, we see

$$S = \frac{f(u_r) - f(u_l)}{u_r - u_l} \tag{4.9}$$

For Burgers' equation, substituting in $f(u) = u^2/2$, we get

$$S = \frac{1}{2}(u_l + u_r) \tag{4.10}$$

With the shock speed known, the Riemann problem is straightforward. If there is a shock (compression, so $u_l > u_r$) then we compute the shock speed and check whether the shock is moving to the left or right, and then use the appropriate state. If there is no shock, then we can simply use upwinding, as there is no ambiguity as to how to trace backwards in time to the correct state. Putting this together, we have:

$$\underset{\text{(shock)}}{\text{if } u_l > u_r:} \qquad u_s = \begin{cases} u_l & \text{if } S > 0 \\ u_r & \text{if } S < 0 \\ 0 & \text{if } S = 0 \end{cases} \tag{4.11}$$

$$\text{otherwise:} \qquad u_s = \begin{cases} u_l & \text{if } u_l > 0 \\ u_r & \text{if } u_r < 0 \\ 0 & \text{otherwise} \end{cases} \tag{4.12}$$

Once the interface states are constructed, the flux is calculated as:

$$F^{n+1/2}_{i+1/2} = \frac{1}{2}\left(u^{n+1/2}_{i+1/2}\right)^2 \tag{4.13}$$

and the conservative update is

$$u^{n+1}_i = u^n_i + \frac{\Delta t}{\Delta x}\left(F^{n+1/2}_{i-1/2} - F^{n+1/2}_{i+1/2}\right) \tag{4.14}$$

The timestep constraint now must consider the most restrictive Courant condition over all the zones:

$$\Delta t = \min_i \{\Delta x/u_i\} \tag{4.15}$$

Figure 4.2 shows the solution for two cases: a rarefaction and a sine-wave steepening into a shock, using a piecewise linear reconstruction and the MC limiter.

Exercise 4.1: *Extend your 1-d finite-volume solver for advection (from Exercise 3.6) to solver Burgers' equation. You will need to change the Riemann solver and use the local velocity in the construction of the interface states. Run the examples shown in Figure 4.2.*

4.2 Going further

- The equation we've been dealing with here is the *inviscid* Burgers' equation. The full Burgers' equation includes viscosity (a velocity diffusion):

$$u_t + uu_x = \epsilon u_{xx} \tag{4.16}$$

  To solve this, we need to first learn about techniques for diffusion, and then how to solve equations with multiple PDE types. This will be described later.

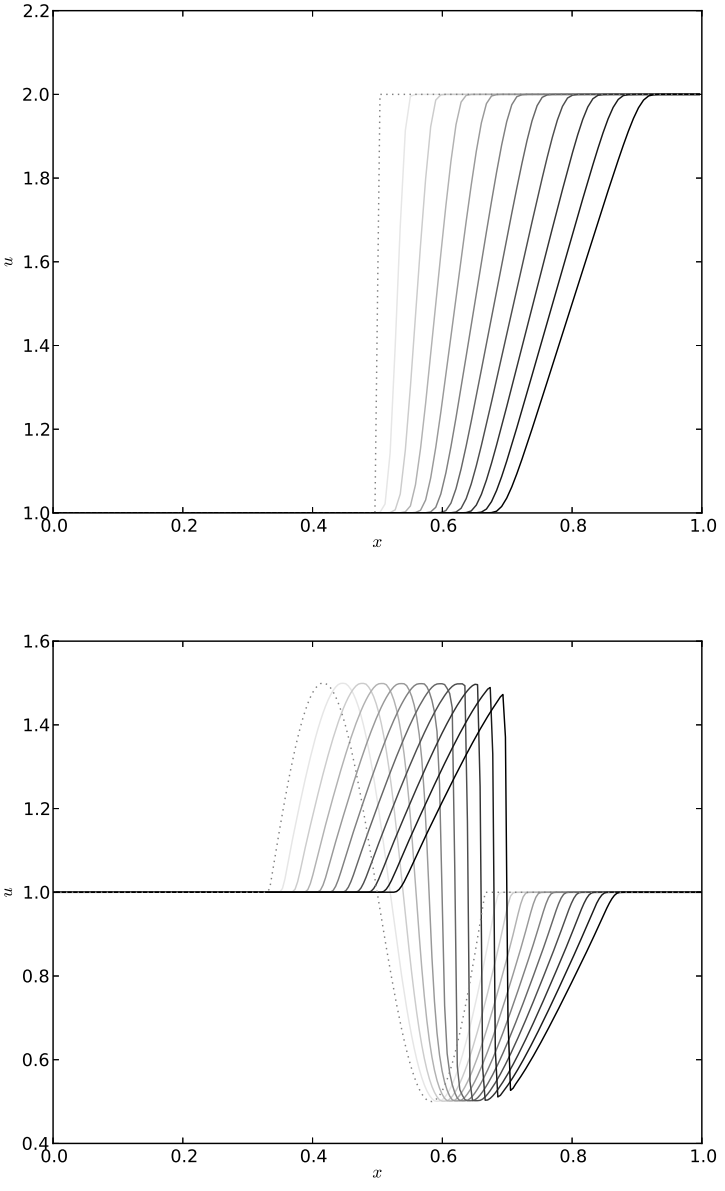

Figure 4.2: Solution to the inviscid Burgers' equation with 256 zones and a Courant number, $C = 0.8$. At the top, is a rarefaction—the left half of the domain was initialized with $u = 1$ and the right half with $u = 2$, creating a divergent flow. On the bottom is a sine-wave steepening into a shock. The curves are shown 0.02 s apart.

hydro_examples: `burgers.py`

Chapter 5

Euler Equations

These notes describe how to do a piecewise linear or piecewise parabolic method for the Euler equations. These are the basis for some of the most popular methods in astrophysics.

5.1 Euler equation properties

The Euler equations in one dimension appear as:

$$\frac{\partial \rho}{\partial t} + \frac{\partial (\rho u)}{\partial x} = 0 \tag{5.1}$$

$$\frac{\partial (\rho u)}{\partial t} + \frac{\partial (\rho u u + p)}{\partial x} = 0 \tag{5.2}$$

$$\frac{\partial (\rho E)}{\partial t} + \frac{\partial (\rho u E + u p)}{\partial x} = 0 \tag{5.3}$$

These represent conservation of mass, momentum, and energy. Here ρ is the density, u is the one-dimensional velocity, p is the pressure, and E is the total energy / mass, and can be expressed in terms of the specific internal energy and kinetic energy as:

$$E = e + \frac{1}{2} u^2 \tag{5.4}$$

The equations are closed with the addition of an equation of state. A common choice is the gamma-law EOS:

$$p = \rho e (\gamma - 1) \tag{5.5}$$

where γ is the ratio of specific heats for the gas/fluid (for an ideal, monatomic gas, $\gamma = 5/3$), but any relation of the form $p = p(\rho, e)$ will work.

One thing that we can notice immediately is that there is no need for temperature in this equation set, although often, when source terms are present, we will need to obtain temperature from the equation of state.

In this form, the equations are said to be in *conservative form*, i.e. they can be written as:

$$U_t + [F(U)]_x = 0 \tag{5.6}$$

with

$$U = \begin{pmatrix} \rho \\ \rho u \\ \rho E \end{pmatrix} \qquad F(U) = \begin{pmatrix} \rho u \\ \rho u u + p \\ \rho u E + u p \end{pmatrix} \tag{5.7}$$

We can write this in *quasi-linear* form by first expressing the flux vector in terms of the conserved variables directly. Taking $u_1 = \rho, u_2 = \rho u, u_3 = \rho E$, we have

$$F(U) = \begin{pmatrix} u_2 \\ \frac{u_2^2}{u_1}\left(1 - \frac{\hat{\gamma}}{2}\right) + u_3\hat{\gamma} \\ \frac{u_3 u_2}{u_1}(1+\hat{\gamma}) - \frac{1}{2}\frac{u_2^3}{u_1^2}\hat{\gamma} \end{pmatrix} \tag{5.8}$$

where $\hat{\gamma} = \gamma - 1$, and $p = \rho e \hat{\gamma} = (\rho E - \rho u^2/2)\hat{\gamma}$. The Jacobian of this flux vector is $A = \partial F/\partial U$:

$$A(U) = \begin{pmatrix} 0 & 1 & 0 \\ -\frac{1}{2}u^2(3-\gamma) & u(3-\gamma) & \gamma - 1 \\ \frac{1}{2}(\gamma-2)u^3 - \frac{uc^2}{\gamma-1} & \frac{3-2\gamma}{2}u^2 + \frac{c^2}{\gamma-1} & u\gamma \end{pmatrix} \tag{5.9}$$

where the speed of sound is $c = \sqrt{\gamma p/\rho}$. With this, our system can be written as:

$$U_t + A(U)U_x = 0 \tag{5.10}$$

This matrix is quite complex and difficult to work with. The eigenvectors of this matrix can be found in a variety of sources (e.g. [51, 48]).

An alternate way to express these equations is using the *primitive variables*: ρ, u, p.

Exercise 5.1: *Show that the Euler equations in primitive form can be written as*

$$q_t + A(q)q_x = 0 \tag{5.11}$$

where

$$q = \begin{pmatrix} \rho \\ u \\ p \end{pmatrix} \qquad A(q) = \begin{pmatrix} u & \rho & 0 \\ 0 & u & 1/\rho \\ 0 & \gamma p & u \end{pmatrix} \tag{5.12}$$

The eigenvalues of A can be found via $|A - \lambda I| = 0$, where $|\ldots|$ indicates the determinant and λ are the eigenvalues.

Exercise 5.2: *Show that the eigenvalues of A are* $\lambda^{(-)} = u - c$, $\lambda^{(\circ)} = u$, $\lambda^{(+)} = u + c$.

Note that both the conserved Jacobian matrix, $A(U)$, and the primitive variable matrix, $A(q)$, have the same eigenvalues—as expected, since they represent the same physics. Also note that in Eq. 5.12, we used the algebraic gamma-law equation of state to replace e with p, however, for a general equation of state, we can get the appropriate expression by writing $p = p(\rho, s)$:

$$\frac{Dp}{Dt} = \left.\frac{\partial p}{\partial \rho}\right|_s \frac{D\rho}{Dt} + \left.\frac{\partial p}{\partial s}\right|_\rho \cancelto{0}{\frac{Ds}{Dt}} \tag{5.13}$$

where Ds/Dt is 0 when no entropy sources are present. Recognizing that $\Gamma_1 \equiv \partial \log p / \partial \log \rho|_s$, we have:

$$\frac{\partial p}{\partial t} + u\frac{\partial p}{\partial x} + \Gamma_1 p \frac{\partial u}{\partial x} = 0 \tag{5.14}$$

as the generalization of the pressure equation.

We'll use the symbols $\{-, \circ, +\}$ to denote the eigenvalues and their corresponding eigenvectors throughout these notes. These eigenvalues are the speeds at which information propagates through the fluid. Since the eigenvalues are real, this system (the Euler equations) is said to be *hyperbolic*. Additionally, since $A = A(q)$, the system is said to be *quasi-linear*. The right and left eigenvectors can be found via:

$$A\, r^{(\nu)} = \lambda^{(\nu)} r^{(\nu)}\,; \qquad l^{(\nu)}\, A = \lambda^{(\nu)} l^{(\nu)} \tag{5.15}$$

where $\nu = \{-, \circ, +\}$ corresponding to the three waves, and there is one right and one left eigenvector for each of the eigenvalues.

Exercise 5.3: *Show that the right eigenvectors are:*

$$r^{(-)} = \begin{pmatrix} 1 \\ -c/\rho \\ c^2 \end{pmatrix} \qquad r^{(\circ)} = \begin{pmatrix} 1 \\ 0 \\ 0 \end{pmatrix} \qquad r^{(+)} = \begin{pmatrix} 1 \\ c/\rho \\ c^2 \end{pmatrix} \tag{5.16}$$

and the left eigenvectors are:

$$l^{(-)} = \begin{pmatrix} 0 & -\frac{\rho}{2c} & \frac{1}{2c^2} \end{pmatrix} \tag{5.17}$$

$$l^{(\circ)} = \begin{pmatrix} 1 & 0 & -\frac{1}{c^2} \end{pmatrix} \tag{5.18}$$

$$l^{(+)} = \begin{pmatrix} 0 & \frac{\rho}{2c} & \frac{1}{2c^2} \end{pmatrix} \tag{5.19}$$

Note that in general, there can be an arbitrary constant in front of each eigenvector. Here they are normalized such that $l^{(i)} \cdot r^{(j)} = \delta_{ij}$.

An IPython notebook using SymPy that derives these is available here: hydro_examples: `euler.ipynb` [1]

A final form of the equations is called the *characteristic form*. Here, we wish to diagonalize the matrix A. We take the matrix R to be the matrix of right eigenvectors, $R = (r^{(-)}|r^{(\circ)}|r^{(+)})$, and L is the corresponding matrix of left eigenvectors:

$$L = \begin{pmatrix} - & l^{(-)} & - \\ - & l^{(\circ)} & - \\ - & l^{(+)} & - \end{pmatrix} \tag{5.20}$$

Note that $L\,R = I = R\,L$, and $L = R^{-1}$.

> **Exercise 5.4:** *Show that $\Lambda = LAR$ is a diagonal matrix with the diagonal elements simply the 3 eigenvalues we found above:*
>
> $$\Lambda = \begin{pmatrix} \lambda^{(-)} & & \\ & \lambda^{(\circ)} & \\ & & \lambda^{(+)} \end{pmatrix} \tag{5.21}$$

Defining $dw = Ldq$, we can write our system as:

$$w_t + \Lambda w_x = 0 \tag{5.22}$$

Here, the w are the characteristic variables. Note that we cannot in general integrate $dw = Ldq$ to write down the characteristic quantities. Since Λ is diagonal, this system is a set of decoupled advection-like equations. If the system were linear, then the solution to each would simply be to advect the quantity $w^{(\nu)}$ at the wave speed $\lambda^{(\nu)}$.

Imagine initial conditions consisting of a jump in the primitive variables, Δq. The corresponding characteristic variables are $\Delta w \sim L\Delta q$ (where the $\sim$ accounts for the fact that in a nonlinear system, $L = L(q)$). The characteristic form of the equations says that each of the waves will carry with it a jump in w. Since $dq = L^{-1}dw = Rdw$, the jump in the primitive variable across each wave is proportional to the right-eigenvector associated with that wave. So, for example, since $r^{(\circ)}$ is only non-zero for the density element, this then means that only density jumps across the $\lambda^{(\circ)} = u$ wave—pressure and velocity are constant across this wave (see for example, Toro [51], Ch. 2, 3 or LeVeque [32] for a thorough discussion). Figure 5.1 shows the three waves emanating from an initial discontinuity.

[1] if you don't have IPython, you can view this rendered online via nbviewer: `http://nbviewer.ipython.org/github/zingale/hydro_examples/blob/master/compressible/euler.ipynb`

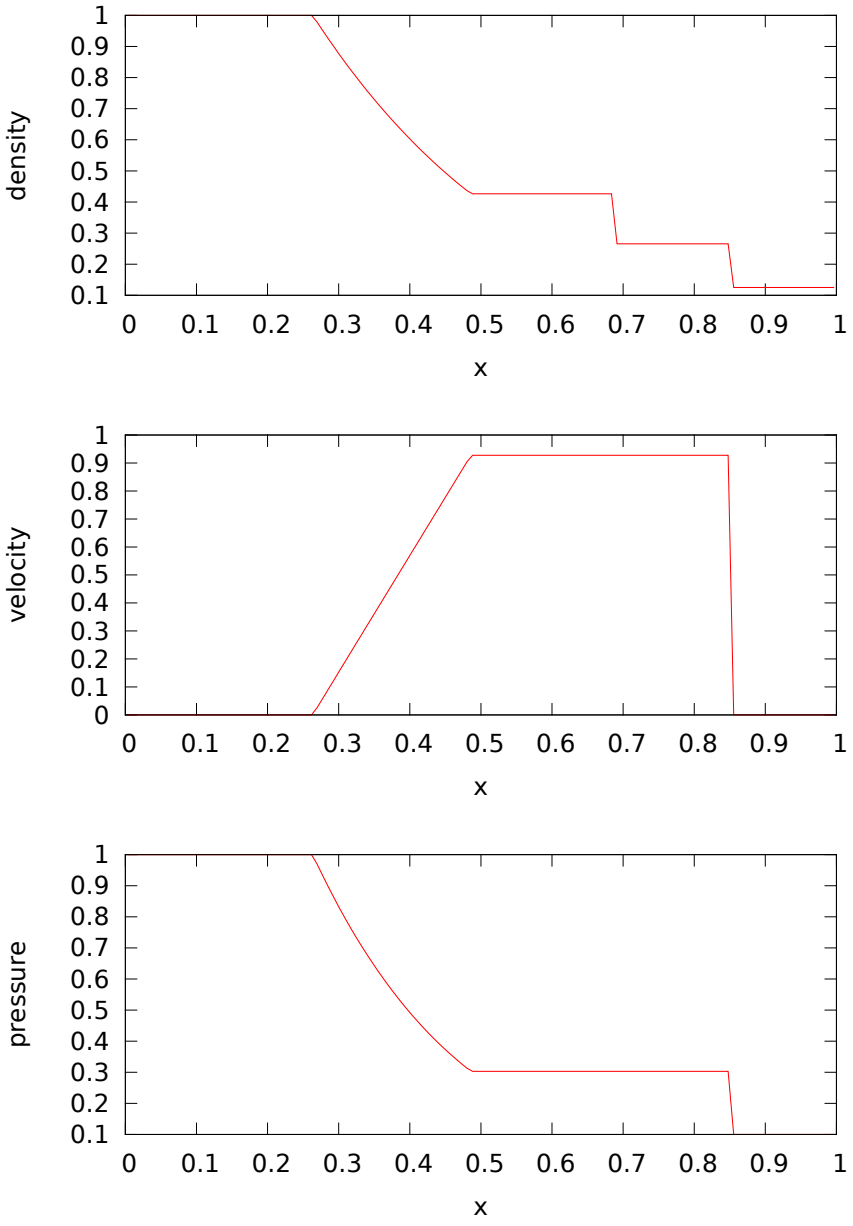

Figure 5.1: Evolution following from an initial discontinuity at $x = 0.5$. These particular conditions are called the *Sod problem*, and in general, a setup with two states separated by a discontinuity is called a shock-tube problem. Here we see the three waves propagating away from the initial discontinuity. The left $(u - c)$ wave is a rarefaction, the middle (u) is the contact discontinuity, and the right $(u + c)$ is a shock. Note that all 3 primitive variables jump across the left and right waves, but only the density jumps across the middle wave. This reflects the right eigenvectors. Also note that no waves have reached the far left and far right yet, the conditions there are the same as the initial conditions.

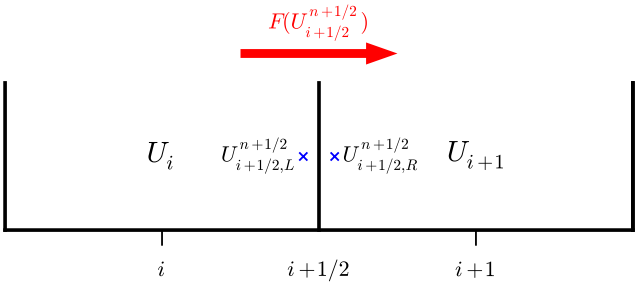

Figure 5.2: The left and right states at interface $i + 1/2$. The arrow indicates the flux through the interface, as computed by the Riemann solver using these states as input.

5.2 Reconstruction of interface states

We will solve the Euler equations using a high-order *Godunov method*—a finite volume method whereby the fluxes through the interfaces are computed by solving the Riemann problem for our system. The finite-volume update for our system appears as:

$$U_i^{n+1} = U_i^n + \frac{\Delta t}{\Delta x}\left(F_{i-1/2}^{n+1/2} - F_{i+1/2}^{n+1/2}\right) \tag{5.23}$$

This says that each of the conserved quantities in U change only due to the flux of that quantity through the boundary of the cell.

Instead of approximating the flux itself on the interface, we find an approximation to the state on the interface, $U_{i-1/2}^{n+1/2}$ and $U_{i+1/2}^{n+1/2}$ and use this with the flux function to define the flux through the interface:

$$F_{i-1/2}^{n+1/2} = F(U_{i-1/2}^{n+1/2}) \tag{5.24}$$
$$F_{i+1/2}^{n+1/2} = F(U_{i+1/2}^{n+1/2}) \tag{5.25}$$

To find this interface state, we predict left and right states at each interface (centered in time), which are the input to the Riemann solver. The Riemann solver will then look at the characteristic wave structure and determine the fluid state on the interface, which is then used to compute the flux. This is illustrated in Figure 5.2.

Finally, although we use the conserved variables for the final update, in constructing the interface states it is often easier to work with the primitive variables. These have a simpler characteristic structure. The interface states in terms of the primitive variables can be converted into the interface states of the conserved variables through a simple algebraic transformation,

$$U_{i+1/2,L}^{n+1/2} = U(q_{i+1/2,L}^{n+1/2}) \tag{5.26}$$

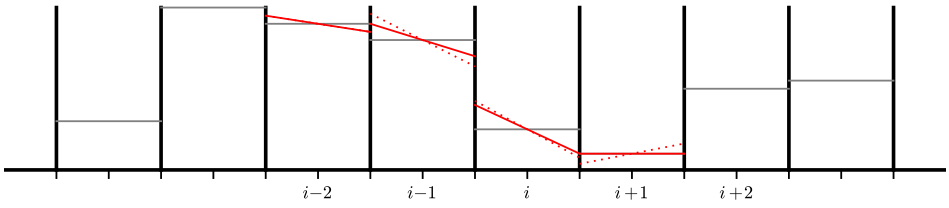

Figure 5.3: Piecewise linear reconstruction of the cell averages. The dotted line shows the unlimited center-difference slopes and the solid line shows the limited slopes.

Constructing these interface states requires reconstructing the cell-average data with a piecewise constant, linear, or parabolic polynomial and doing characteristic tracing to see how much of each characteristic quantity comes to the interface over the timestep. The jump in the primitive variables is projected into the characteristic variables, and only jumps moving toward the interface are included in our reconstruction. We look at several methods below that build off of these ideas below.

5.2.1 Piecewise constant

The simplest possible reconstruction of the data is piecewise constant. This is what was done in the original Godunov method. For the interface marked by $i + 1/2$, the left and right states on the interface are simply:

$$U_{i+1/2,L} = U_i \tag{5.27}$$
$$U_{i+1/2,R} = U_{i+1} \tag{5.28}$$

This does not take into account in any way how the state U may be changing through the cell. As a result, it is first-order accurate in space, and since no attempt was made to center it in time, it is first-order accurate in time.

5.2.2 Piecewise linear

For higher-order reconstruction, we first convert from the conserved variables, U, to the primitive variables, q. These have a simpler characteristic structure, making them easier to work with. Here we consider piecewise linear reconstruction—the cell average data is approximated by a line with non-zero slope within each cell. Figure 5.3 shows the piecewise linear reconstruction of some data.

Consider constructing the left state at the interface $i + 1/2$ (see Figure 5.2). Just like for the advection equation, we do a Taylor expansion through $\Delta x/2$ to bring us to the interface, and $\Delta t/2$ to bring us to the midpoint in time. Starting with q_i,

the cell-centered primitive variable, expanding to the right interface (to create the left state there) gives:

$$q_{i+1/2,L}^{n+1/2} = q_i^n + \frac{\Delta x}{2}\frac{\partial q}{\partial x}\bigg|_i + \frac{\Delta t}{2}\underbrace{\frac{\partial q}{\partial t}\bigg|_i}_{=-A\partial q/\partial x} + \ldots \tag{5.29}$$

$$= q_i^n + \frac{\Delta x}{2}\frac{\partial q}{\partial x}\bigg|_i - \frac{\Delta t}{2}\left(A\frac{\partial q}{\partial x}\right)_i \tag{5.30}$$

$$= q_i^n + \frac{1}{2}\left[1 - \frac{\Delta t}{\Delta x}A_i\right]\Delta q_i \tag{5.31}$$

where Δq_i is the reconstructed slope of the primitive variable in that cell (similar to how we compute it for the advection equation). We note that the terms truncated in the first line are $O(\Delta x^2)$ and $O(\Delta t^2)$, so our method will be second-order accurate in space and time.

As with the advection equation, we limit the slope such that no new minima or maxima are introduced. Any of the slope limiters used for linear advection apply here as well. We represent the limited slope as $\overline{\Delta q}_i$.

We can decompose $A\Delta q$ in terms of the left and right eigenvectors and sum over all the waves that move *toward* the interface. First, we recognize that $A = R\Lambda L$ and recognizing that the '1' in Eq. 5.31 is the identity, $I = LR$, we rewrite this expression as:

$$q_{i+1/2,L}^{n+1/2} = q_i^n + \frac{1}{2}\left[RL - \frac{\Delta t}{\Delta x}R\Lambda L\right]_i \overline{\Delta q}_i \tag{5.32}$$

We see the common factor of $L\Delta q$. We now write this back in component form. Consider:

$$R\Lambda L\overline{\Delta q} = \left(\begin{array}{c:c:c} r_1^{(-)} & r_1^{(\circ)} & r_1^{(+)} \\ r_2^{(-)} & r_2^{(\circ)} & r_2^{(+)} \\ r_3^{(-)} & r_3^{(\circ)} & r_3^{(+)} \end{array}\right)\left(\begin{array}{ccc} \lambda^{(-)} & & \\ & \lambda^{(\circ)} & \\ & & \lambda^{(+)} \end{array}\right)\left(\begin{array}{ccc} l_1^{(-)} & l_2^{(-)} & l_3^{(-)} \\ \hdashline l_1^{(\circ)} & l_2^{(\circ)} & l_3^{(\circ)} \\ \hdashline l_1^{(+)} & l_2^{(+)} & l_3^{(+)} \end{array}\right)\left(\begin{array}{c} \overline{\Delta\rho} \\ \overline{\Delta u} \\ \overline{\Delta p} \end{array}\right) \tag{5.33}$$

Starting with $L\overline{\Delta q}$, which is a vector with each component the dot-product of a left eigenvalue with $\overline{\Delta q}$, we have

$$R\Lambda L\overline{\Delta q} = \left(\begin{array}{c:c:c} r_1^{(-)} & r_1^{(\circ)} & r_1^{(+)} \\ r_2^{(-)} & r_2^{(\circ)} & r_2^{(+)} \\ r_3^{(-)} & r_3^{(\circ)} & r_3^{(+)} \end{array}\right)\left(\begin{array}{ccc} \lambda^{(-)} & & \\ & \lambda^{(\circ)} & \\ & & \lambda^{(+)} \end{array}\right)\left(\begin{array}{c} l^{(-)}\cdot\overline{\Delta q} \\ l^{(\circ)}\cdot\overline{\Delta q} \\ l^{(+)}\cdot\overline{\Delta q} \end{array}\right) \tag{5.34}$$

Next we see that multiplying this vector by Λ simply puts the eigenvalue with its

respective eigenvector in the resulting column vector:

$$R\Lambda L\overline{\Delta q} = \left(\begin{array}{c|c|c} r_1^{(-)} & r_1^{(\circ)} & r_1^{(+)} \\ r_2^{(-)} & r_2^{(\circ)} & r_2^{(+)} \\ r_3^{(-)} & r_3^{(\circ)} & r_3^{(+)} \end{array}\right) \left(\begin{array}{c} \lambda^{(-)}\, l^{(-)} \cdot \overline{\Delta q} \\ \lambda^{(\circ)}\, l^{(\circ)} \cdot \overline{\Delta q} \\ \lambda^{(+)}\, l^{(+)} \cdot \overline{\Delta q} \end{array}\right) \tag{5.35}$$

Finally, the last multiply results in a column vector:

$$R\Lambda L\overline{\Delta q} = \left(\begin{array}{c} r_1^{(-)}\,\lambda^{(-)}\, l^{(-)} \cdot \overline{\Delta q} + r_1^{(\circ)}\,\lambda^{(\circ)}\, l^{(\circ)} \cdot \overline{\Delta q} + r_1^{(+)}\,\lambda^{(+)}\, l^{(+)} \cdot \overline{\Delta q} \\ r_2^{(-)}\,\lambda^{(-)}\, l^{(-)} \cdot \overline{\Delta q} + r_2^{(\circ)}\,\lambda^{(\circ)}\, l^{(\circ)} \cdot \overline{\Delta q} + r_2^{(+)}\,\lambda^{(+)}\, l^{(+)} \cdot \overline{\Delta q} \\ r_3^{(-)}\,\lambda^{(-)}\, l^{(-)} \cdot \overline{\Delta q} + r_3^{(\circ)}\,\lambda^{(\circ)}\, l^{(\circ)} \cdot \overline{\Delta q} + r_3^{(+)}\,\lambda^{(+)}\, l^{(+)} \cdot \overline{\Delta q} \end{array}\right) \tag{5.36}$$

We can rewrite this compactly as:

$$\sum_\nu \lambda^{(\nu)} (l^{(\nu)} \cdot \overline{\Delta q}) r^{(\nu)} \tag{5.37}$$

where we use ν to indicate which wave we are summing over. A similar expansion is used for $RL\overline{\Delta q}$. In fact, any vector can be decomposed in this fashion:

$$\chi = I\chi = RL\chi = \sum_\nu (l^{(\nu)} \cdot \chi) r^{(\nu)} \tag{5.38}$$

And then it is easy to see that the above manipulations for $A\Delta q$ can be expressed as:

$$A\Delta q = A\sum_\nu (l^{(\nu)} \cdot \overline{\Delta q}) r^{(\nu)} = \sum_\nu (l^{(\nu)} \cdot \overline{\Delta q}) A r^{(\nu)} = \sum_\nu (l^{(\nu)} \cdot \overline{\Delta q}) \lambda^{(\nu)} r^{(\nu)} \tag{5.39}$$

where we used $Ar^{(\nu)} = \lambda^{(\nu)} r^{(\nu)}$. The quantity $(l^{(\nu)} \cdot \overline{\Delta q})$ that shows up here is the projection of the vector $\overline{\Delta q}$ into the characteristic variable carried by wave ν. This sum shows, as discussed earlier, that each wave carries a jump in the characteristic variable, with the jump in the primitive variables proportion to the right eigenvector, $r^{(\nu)}$.

The resulting vector for the left state is:

$$q_{i+1/2,L}^{n+1/2} = q_i^n + \frac{1}{2} \sum_{\nu;\lambda^{(\nu)} \geq 0} \left[1 - \frac{\Delta t}{\Delta x} \lambda_i^{(\nu)}\right] (l_i^{(\nu)} \cdot \overline{\Delta q}_i) r_i^{(\nu)} \tag{5.40}$$

Note that we make a slight change here, and only include a term in the sum if its wave is moving toward the interface ($\lambda^{(\nu)} \geq 0$). The quantity $\Delta t \lambda^{(\nu)}/\Delta x$ inside the brackets is simply the CFL number for the wave ν.

Starting with the data in the $i+1$ zone and expanding to the left, we can find the right state on the $i+1/2$ interface:

$$q_{i+1/2,R}^{n+1/2} = q_{i+1}^n - \frac{1}{2} \sum_{\nu;\lambda^{(\nu)} \leq 0} \left[1 + \frac{\Delta t}{\Delta x} \lambda_{i+1}^{(\nu)}\right] (l_{i+1}^{(\nu)} \cdot \overline{\Delta q}_{i+1}) r_{i+1}^{(\nu)} \tag{5.41}$$

A good discussion of this is in Miller & Colella [37] (Eq. 85). This expression is saying that each wave carries a jump in $r^{(\nu)}$ and only those jumps moving toward the interface contribute to our interface state. This restriction of only summing up the waves moving toward the interface is sometimes called *characteristic tracing*. This decomposition in terms of the eigenvectors and eigenvalues is commonly called a *characteristic projection*. In terms of an operator, P, it can be expressed as:

$$P\chi = \sum_{\nu} (l^{(\nu)} \cdot \chi) r^{(\nu)} \tag{5.42}$$

Exercise 5.5: *Show that $Pq = q$, using the eigenvectors corresponding to the primitive variable form of the Euler equations.*

In the literature, sometimes a '$>$' or '$<$' subscript on P is used to indicate the characteristic tracing.

We could stop here, but Colella & Glaz [20] (p. 278) argue that the act of decomposing A in terms of the left and right eigenvectors is a linearization of the quasi-linear system, and we should minimize the size of the quantities that are subjected to this characteristic projection. To accomplish this, they suggest subtracting off a *reference state*. Saltzman (Eq. 8) further argues that since only jumps in the solution are used in constructing the interface state, and that the characteristic decomposition simply adds up all these jumps, we can subtract off the reference state and project the result. In other words, we can write:

$$q^{n+1/2}_{i+1/2,L} - q_{\text{ref}} = q^n_i - q_{\text{ref}} + \frac{1}{2}\left[1 - \frac{\Delta t}{\Delta x} A_i\right] \overline{\Delta q}_i \tag{5.43}$$

Then we subject the RHS to the characteristic projection—this tells us how much of the quantity $q^{n+1/2}_{i+1/2,L} - q_{\text{ref}}$ reaches the interface. Colella & Glaz (p. 278) and Colella (Eq. 2.11) suggest

$$q_{\text{ref}} = \tilde{q}_{i,L} \equiv q_i + \frac{1}{2}\left[1 - \frac{\Delta t}{\Delta x} \max(\lambda^{(+)}_i, 0)\right] \overline{\Delta q}_i \tag{5.44}$$

where $\lambda^{(+)}$ is the fastest eigenvalue, and thus will see the largest portion of the linear profiles. Physically, this reference state represents the jump carried by the fastest wave moving toward the interface. Then,

$$q^{n+1/2}_{i+1/2,L} - \tilde{q}_{i,L} = \frac{1}{2}\frac{\Delta t}{\Delta x}\left[\max(\lambda^{(+)}_i, 0) - A_i\right] \overline{\Delta q}_i \tag{5.45}$$

and projecting this RHS (see Colella & Glaz Eq. 43; Miller & Colella Eq. 87), and isolating the interface state, we have

$$q^{n+1/2}_{i+1/2,L} = \tilde{q}_{i,L} + \frac{1}{2}\frac{\Delta t}{\Delta x} \sum_{\nu;\lambda^{(\nu)}\geq 0} l^{(\nu)}_i \cdot \left[\max(\lambda^{(+)}_i, 0) - A_i\right] \overline{\Delta q}_i \, r^{(\nu)}_i \tag{5.46}$$

$$= \tilde{q}_{i,L} + \frac{1}{2}\frac{\Delta t}{\Delta x} \sum_{\nu;\lambda^{(\nu)}\geq 0} \left[\max(\lambda^{(+)}_i, 0) - \lambda^{(\nu)}_i\right] (l^{(\nu)}_i \cdot \overline{\Delta q}_i) \, r^{(\nu)}_i \tag{5.47}$$

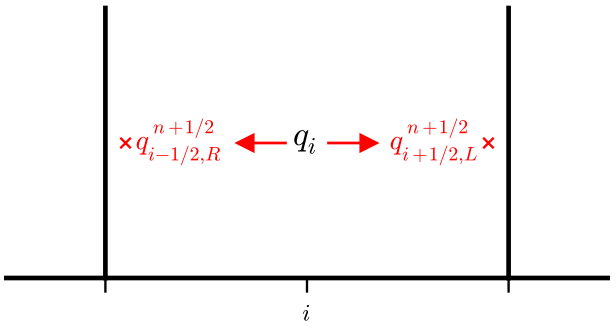

Figure 5.4: The two interface states that are constructed using q_i as the starting point.

This is equivalent to the expression in Saltzman [45] (p. 161, first column, second-to-last equation) and Colella [19] (p. 191, the group of expressions at the end). The corresponding state to the right of this interface is:

$$q^{n+1/2}_{i+1/2,R} = \tilde{q}_{i+1,R} + \frac{1}{2}\frac{\Delta t}{\Delta x} \sum_{\nu;\lambda^{(\nu)}\leq 0} \left[\min(\lambda^{(-)}_{i+1}, 0) - \lambda^{(\nu)}_{i+1}\right] (l^{(\nu)}_{i+1} \cdot \overline{\Delta q}_{i+1})\, r^{(\nu)}_{i+1} \tag{5.48}$$

where now the reference state captures the flow from the $i+1$ zone moving to the *left* to this interface (hence the appearance of $\lambda^{(-)}$, the leftmost eigenvalue):

$$\tilde{q}_{i+1,R} = q_{i+1} - \frac{1}{2}\left[1 + \frac{\Delta t}{\Delta x}\min(\lambda^{(-)}_{i+1}, 0)\right] \overline{\Delta q}_{i+1} \tag{5.49}$$

Side note: the data in zone i will be used to construct the right state at $i-1/2$ (the left interface) and the left state at $i+1/2$ (the right interface) (see Figure 5.4). For this reason, codes usually compute the eigenvectors/eigenvalues for that zone and then compute $q^{n+1/2}_{i-1/2,R}$ together with $q^{n+1/2}_{i+1/2,L}$ in a loop over the zone centers.

5.2.3 Piecewise parabolic

The piecewise parabolic method uses a parabolic reconstruction in each cell. This is more accurate than the linear reconstruction. Figure 5.5 shows the reconstructed parabolic profiles within a few cells. Since the original PPM paper [23], there have been many discussions of the method, with many variations. Here we focus on the presentation by Miller & Colella [37], since that is the most straightforward. Note: even though a parabolic profile could be third-order accurate, the temporal discretization and prediction in this method is still only second-order.

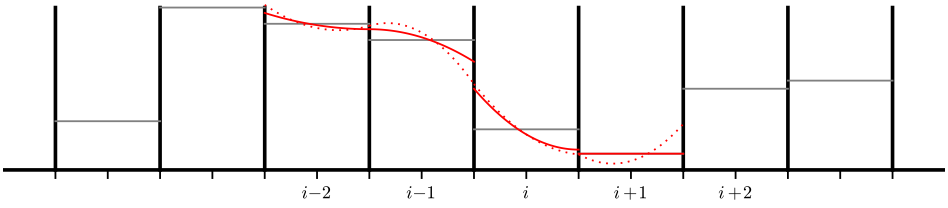

Figure 5.5: Piecewise parabolic reconstruction of the cell averages. The dotted line shows the unlimited parabolas—note how they touch at each interface, since the interface values come from the same interpolant initially. The solid line shows the limited parabolas.

Miller & Colella give an excellent description of how to take the results for piecewise linear reconstruction and generalize it to the case of PPM [23] (see Eqs. 88-90). Starting with Eq. 5.43, we can write this (after the characteristic projection) as

$$q^{n+1/2}_{i+1/2,L} = \tilde{q}_+ - \sum_{\nu;\lambda^{(\nu)}\geq 0} l_i^{(\nu)} \cdot \left\{ \tilde{q}_+ - \left[q_i^n + \frac{1}{2}\left(1 - \frac{\Delta t}{\Delta x}\lambda_i^{(\nu)}\right)\overline{\Delta q}_i \right] \right\} r_i^{(\nu)} \tag{5.50}$$

Miller & Colella rewrite the portion inside the $[\ldots]$ recognizing that (similar to M&C Eq. 88, but for the $i+1/2, L$ interface):

$$q_i^n + \frac{1}{2}\left(1 - \frac{\Delta t}{\Delta x}\lambda_i^{(\nu)}\right)\overline{\Delta q}_i \approx \frac{1}{\lambda \Delta t}\int_{x_{i+1/2}-\lambda\Delta t}^{x_{i+1/2}} q(x)dx \tag{5.51}$$

where $q(x)$ is the reconstructed functional form of q in the zone.

> **Exercise 5.6:** *Show that this is exactly true for a linear reconstruction of $q(x)$, i.e., $q(x) = q_i + (\partial q/\partial x)(x - x_i)$.*

The integral on the right represents the average of q that can reach the right interface of the cell i over timestep Δt, moving at the wavespeed λ. This suggests that we can replace the linear reconstruction of q with a parabolic one, and keep our expressions for the interface states.

In particular, we define

$$\mathcal{I}_+^{(\nu)}(q_i) = \frac{1}{\sigma^{(\nu)}\Delta x}\int_{x_{i+1/2}-\sigma^{(\nu)}\Delta x}^{x_{i+1/2}} q(x)dx \tag{5.52}$$

with $\sigma^{(\nu)} = |\lambda^{(\nu)}|\Delta t/\Delta x$ (see Almgren et al. Eq. 31) (see Figure 5.6). Then

$$q^{n+1/2}_{i+1/2,L} = \tilde{q}_+ - \sum_{\nu;\lambda^{(\nu)}\geq 0} l_i^{(\nu)} \cdot \left(\tilde{q}_+ - \mathcal{I}_+^{(\nu)}(q_i)\right) r_i^{(\nu)} \tag{5.53}$$

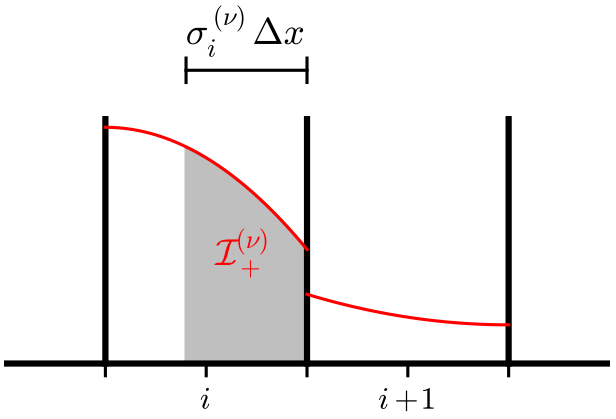

Figure 5.6: Integration under the parabolic profile. For each of the waves, $\sigma^{(\nu)}$ is the fraction of the cell that they cross in a timestep, and $\sigma^{(\nu)}\Delta x = \lambda^{(\nu)}\Delta t$ is the distance they can travel. Here we are integrating under the parabola to the right interface of cell i to define $\mathcal{I}_+^{(\nu)}$ (this is indicated by the shaded region). The $\mathcal{I}_+^{(\nu)}$ carried by this wave will be added to those carried by the other waves to form the left state at interface $i+1/2$.

Miller & Colella choose the reference state as

$$\tilde{q}_+ = \begin{cases} \mathcal{I}_+^{(+)}(q_i) & \text{if } u+c>0 \\ q_i & \text{otherwise} \end{cases} \tag{5.54}$$

where the superscript $(+)$ on $\mathcal{I}$ indicates that the fastest eigenvalue ($\lambda^{(+)} = u+c$) is used. This is similar in spirit to Eq. 5.44. Note: in the original PPM paper, if the wave is not approaching the interface, instead of using the cell-average, q_i, they use the limit of the quadratic interpolant. In contrast to the above, the Castro paper [2] just uses q_i for the reference state regardless of whether the wave is moving toward or away from the interface. Note that if the system were linear, then the choice of reference state would not matter.

To finish the reconstruction, we need to know the parabolic form of $q(x)$. Here, we do the reconstruction from the original PPM paper:

$$q(x) = q_- + \xi(x)\left(\Delta q + q_6(1-\xi(x))\right) \tag{5.55}$$

with $\Delta q = q_+ - q_-$, and q_-, q_+ the values of the polynomial on the left and right edges, respectively, of the current cell, and

$$q_6 \equiv 6\left[q_i - \frac{1}{2}(q_- + q_+)\right] \tag{5.56}$$

and

$$\xi(x) = \frac{x - x_{i-1/2}}{\Delta x} \tag{5.57}$$

To complete the description, we need to determine the parameters of the parabola. The values of q_- and q_+ are computed and limited as described in the original PPM paper. With this definition, we can do the integral $\mathcal{I}_+$:

$$\mathcal{I}_+^{(\nu)}(q_i) = q_{+,i} - \frac{\sigma_i^{(\nu)}}{2}\left[\Delta q_i - q_{6,i}\left(1 - \frac{2}{3}\sigma_i^{(\nu)}\right)\right] \tag{5.58}$$

Figure 5.6 illustrates the process of integrating under the parabolic profile.

Exercise 5.7: *Show that $q(x)$ is a conservative interpolant. That is*

$$\frac{1}{\Delta x}\int_{x_{i-1/2}}^{x_{i+1/2}} q(x)dx = q_i \tag{5.59}$$

You can also see that the average over the left half of the zone is $q_i - \frac{1}{4}\Delta q$ and the average over the right half of the zone is $q_i + \frac{1}{4}\Delta q$. This means that there are equal areas between the integral and zone average on the left and right sides of the zone. This can be seen by looking at Figure 5.5.

Aside: Note that this characteristic projection of $\tilde{q}_+ - \mathcal{I}_+^{(\nu)}$ is discussed in the original PPM paper in the paragraph following Eq. 3.5. They do not keep things in this form however, and instead explicitly multiply out the $l \cdot [\ldots] r$ terms to arrive at Eq. 3.6. For example, starting with Eq. 5.53, we can write the left velocity state as (leaving off the i subscripts on the vectors):

$$u_{i+1/2,L}^{n+1/2} = \tilde{u}_+ - \sum_\nu l^{(\nu)} \cdot (\tilde{q}_+ - \mathcal{I}_+^{(\nu)}(q)) \underbrace{r^{(\nu)}}_{\substack{\text{only the} \\ u \text{ 'slot'}}} \tag{5.60}$$

(where, as above, the $\sim$ indicates the reference state). Here the r eigenvector on the end is representative—we only pick the row corresponding to u in the q vector (in our case, the second row).

Putting in the eigenvectors and writing out the sum, we have:

$$\begin{aligned} u_{i+1/2,L}^{n+1/2} = \tilde{u}_+ &- \begin{pmatrix} 0 & -\frac{\rho}{2c} & \frac{1}{2c^2} \end{pmatrix} \begin{pmatrix} \tilde{\rho}_+ - \mathcal{I}_+^{(-)}(\rho) \\ \tilde{u}_+ - \mathcal{I}_+^{(-)}(u) \\ \tilde{p}_+ - \mathcal{I}_+^{(-)}(p) \end{pmatrix} \begin{pmatrix} 1 \\ -c/\rho \\ c^2 \end{pmatrix} \\ &- \begin{pmatrix} 1 & 0 & -\frac{1}{c^2} \end{pmatrix} \begin{pmatrix} \tilde{\rho}_+ - \mathcal{I}_+^{(\circ)}(\rho) \\ \tilde{u}_+ - \mathcal{I}_+^{(\circ)}(u) \\ \tilde{p}_+ - \mathcal{I}_+^{(\circ)}(p) \end{pmatrix} \begin{pmatrix} 1 \\ 0 \\ 0 \end{pmatrix} \\ &- \begin{pmatrix} 0 & \frac{\rho}{2c} & \frac{1}{2c^2} \end{pmatrix} \begin{pmatrix} \tilde{\rho}_+ - \mathcal{I}_+^{(+)}(\rho) \\ \tilde{u}_+ - \mathcal{I}_+^{(+)}(u) \\ \tilde{p}_+ - \mathcal{I}_+^{(+)}(p) \end{pmatrix} \begin{pmatrix} 1 \\ c/\rho \\ c^2 \end{pmatrix} \end{aligned} \tag{5.61}$$

Here again we show the entire right eigenvector for illustration, but only the element that comes into play is drawn in black. This shows that the second term is 0—the contact wave does not carry a jump in velocity. Multiplying out $l^{(\nu)} \cdot (\tilde{q}_+ - \mathcal{I}_+^{(\nu)})$ we have:

$$u_{i+1/2,L}^{n+1/2} = \tilde{u}_+ - \frac{1}{2}\left[(\tilde{u}_+ - \mathcal{I}_+^{(-)}(u)) - \frac{\tilde{p}_+ - \mathcal{I}_+^{(-)}(p)}{C}\right] - \frac{1}{2}\left[(\tilde{u}_+ - \mathcal{I}_+^{(+)}(u)) + \frac{\tilde{p}_+ - \mathcal{I}_+^{(+)}(p)}{C}\right] \tag{5.62}$$

where C is the Lagrangian sound speed ($C = \sqrt{\gamma p \rho}$). Defining

$$\beta^+ = -\frac{1}{2C}\left[(\tilde{u}_+ - \mathcal{I}_+^{(+)}(u)) + \frac{\tilde{p}_+ - \mathcal{I}_+^{(+)}(p)}{C}\right] \tag{5.63}$$

$$\beta^- = +\frac{1}{2C}\left[(\tilde{u}_+ - \mathcal{I}_+^{(-)}(u)) - \frac{\tilde{p}_+ - \mathcal{I}_+^{(-)}(p)}{C}\right] \tag{5.64}$$

we can write our left state as:

$$u_{i+1/2,L}^{n+1/2} = \tilde{u}_+ + C(\beta^+ - \beta^-) \tag{5.65}$$

This is Eqs. 3.6 and 3.7 in the PPM paper. Note that in their construction appears to use the reference state in defining the Lagrangian sound speed (in their β expressions is written as $\tilde{C}$). This may follow from the comment before Eq. 3.6, "modified slightly for the present application". Similarly, the expressions for ρ_L and p_L can be written out.

Similar expressions can be derived for the right state at the left interface of the zone ($q_{i-1/2,R}^{n+1/2}$). Here, the integral under the parabolic reconstruction is done over the region of each wave that can reach the left interface over our timestep:

$$\mathcal{I}_-^{(\nu)}(q) = \frac{1}{\sigma^{(\nu)}\Delta x}\int_{x_{i-1/2}}^{x_{i-1/2}+\sigma^{(\nu)}\Delta x} q(x)dx \tag{5.66}$$

The right state at $i - 1/2$ using zone i data is:

$$q_{i-1/2,R}^{n+1/2} = \tilde{q}_- - \sum_{\nu;\lambda_\nu \le 0} l_i^{(\nu)} \cdot \left(\tilde{q}_- - \mathcal{I}_-^{(\nu)}(q_i)\right) r_i^{(\nu)} \tag{5.67}$$

where the reference state is now:

$$\tilde{q}_- = \begin{cases} \mathcal{I}_-^{(-)}(q_i) & \text{if } u - c < 0 \\ q_i & \text{otherwise} \end{cases} \tag{5.68}$$

where the $(-)$ superscript on $\mathcal{I}$ indicates that the most negative eigenvalue ($\lambda^- = u - c$) is used. The integral $\mathcal{I}_-^{(\nu)}(q)$ can be computed analytically by substituting in the parabolic interpolant, giving:

$$\mathcal{I}_-^{(\nu)}(q_i) = q_{-,i} + \frac{\sigma_i^{(\nu)}}{2}\left[\Delta q_i + q_{6,i}\left(1 - \frac{2}{3}\sigma_i^{(\nu)}\right)\right] \tag{5.69}$$

This is equivalent to Eq. 31b in the Castro paper.

New PPM limiters

Recent work [22] has formulated improved limiters for PPM that do not clip the profiles at extrema. This only changes the limiting process in defining q_+ and q_i, and does not affect the subsequent parts of the algorithm.

5.3 The Riemann problem

Once the interface states are created, the Riemann solver is called. This returns the solution at the interface:

$$q_{i+1/2}^{n+1/2} = \mathcal{R}(q_{i+1/2,L}^{n+1/2}, q_{i+1/2,R}^{n+1/2}) \tag{5.70}$$

Solving the Riemann problem for the Euler equations can be a complex operation, but the general ideas are straightforward. Here we review the basic outline of operations, and refer to Toro [51] for full details on a variety of methods for solving the Riemann problem.

The Riemann problem consists of a left and right state separated by an interface. For the Euler equations, there are three eigenvalues, which are the speeds at which information propagates. Each of these correspond to a wave that will move out from the interface with time, and each wave will carry with it a jump in the characteristic variables. The figure below shows the three waves moving out from the interface, separating space into 4 regions, marked: L, L^*, R^*, and R. We typically work in terms of primitive variables. The states in the L and R regions are simply the left and right input states—the waves have not had time to reach here, so they are unmodified.

The left and right states are connected to the state in the star region by a Hugoniot curve—this is a curve in the u-p plane that shows all of the possible states one can reach from the current state through either a shock or rarefaction. There are two such curves, one corresponding to the left and one to the right state, and the solution to the Riemann problem is the point in the u-p plane where these two curves intersect. Figure 5.8 shows the Hugoniot curves for the Sod problem.

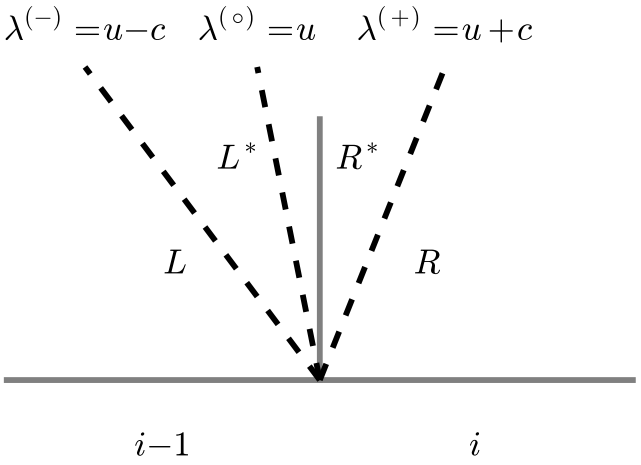

Figure 5.7: The wave structure and 4 distinct regions for the Riemann problem. Time can be thought of as the vertical axis here, so we see the waves moving outward from the interface.

We are interested in the state at the interface. To determine this, we need to determine which region we are in. That requires an estimation of the wave speeds. Since these are nonlinear waves, we cannot in general just use the eigenvalues (although some approximate solvers do). Different Riemann solvers will have different approximations for finding the speeds of the left, center, and right wave. Note the figure shows only one possible configuration for the waves—they can all be on one side of the interface (for all supersonic waves), or the contact (the middle wave) can be on either side.

Once the wave speeds are known, we look at the sign of the speeds to determine which of the 4 regions is on the interface. In the 'star' region, only ρ jumps across the middle (contact) wave, the pressure and velocity are constant across that wave (see $r^{(\circ)}$). We determine the state in the star region ($\rho_l^*, \rho_r^*, u^*, p^*$) by using the jump conditions for the Euler equations. In general, these differ depending on whether the waves are shocks or rarefactions. In practice, approximate Riemann solvers often assume one or the other (for example, the two-shock Riemann solver used in [20]). With the wave speeds and the states known in each region, we can evaluate the state on the interface, $q_{i+1/2}^{n+1/2}$.

Recall that a rarefaction involves diverging flow—it spreads out with time. Special consideration needs to be taken if the rarefaction wave spans the interface (a *transonic rarefaction*). In this case, most Riemann solvers interpolate between the left or right state and the appropriate star state.

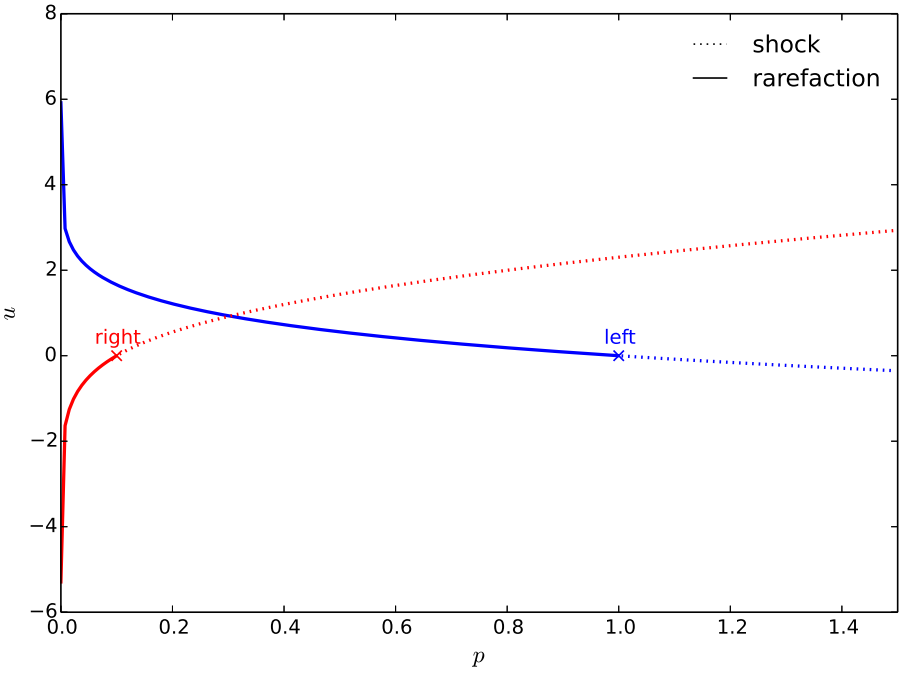

Figure 5.8: The Hugoniot curves corresponding to the Sod problem. The shock and rarefaction curves are shown. The solution to the Riemann problem is the point where the curves intersect.
hydro_examples: `riemann-phase.py`

Then the fluxes are computed from this state as:

$$F^{n+1/2}_{i+1/2} = \begin{pmatrix} \rho^{n+1/2}_{i+1/2} u^{n+1/2}_{i+1/2} \\ \rho^{n+1/2}_{i+1/2} (u^{n+1/2}_{i+1/2})^2 + p^{n+1/2}_{i+1/2} \\ u^{n+1/2}_{i+1/2} p^{n+1/2}_{i+1/2} / (\gamma - 1) + \frac{1}{2} \rho^{n+1/2}_{i+1/2} (u^{n+1/2}_{i+1/2})^3 + u^{n+1/2}_{i+1/2} p^{n+1/2}_{i+1/2} \end{pmatrix} \tag{5.71}$$

Note that instead of returning an approximate state at the interface, some Riemann solvers (e.g. the HLL(C) solvers) instead approximate the fluxes directly. These use estimates of the wave speeds together with the Rankine-Hugoniot jump conditions to give the fluxes.

5.4 Conservative update

Once we have the fluxes, the conservative update is done as

$$U^{n+1}_i = U^n_i + \frac{\Delta t}{\Delta x} \left(F^{n+1/2}_{i-1/2} - F^{n+1/2}_{i+1/2} \right) \tag{5.72}$$

The timestep, Δt is determined by the time it takes for the fastest wave to cross a single zone:

$$\Delta t < \max_i \frac{\Delta x}{|u_i| + c} \tag{5.73}$$

5.5 Other Thermodynamic Equations

At times we will want to use alternate forms of the energy equation. The internal energy is governed by the first law of thermodynamics. In the absence of any heat sources, we have:

$$dq = 0 = de + pd(1/\rho) \tag{5.74}$$

where e is the specific internal energy. Applying this to a Lagrangian fluid element, we have:

$$\frac{De}{Dt} + p\frac{D(1/\rho)}{Dt} = 0 \tag{5.75}$$

$$\frac{De}{Dt} - \frac{1}{\rho^2}\frac{D\rho}{Dt} = 0 \tag{5.76}$$

$$\rho\frac{De}{Dt} + p\nabla \cdot U = 0 \tag{5.77}$$

where we used the continuity equation in the last step to eliminate $D\rho/Dt$. This can be rewritten by adding $e\times$ the continuity equation to give:

$$\frac{\partial(\rho e)}{\partial t} + \nabla \cdot (\rho U e) + p\nabla \cdot U = 0 \tag{5.78}$$

5.5.1 Eigensystem with Temperature

Here we illustrate how the eigensystem changes when we replace pressure with temperature in our primitive variable system.

We write this set of variables as $\hat{q} = (\tau, u, T)^\intercal$—note that we keep τ instead of ρ here. The motivation for this comes from the fact that with temperature in the mix, the temperature will jump across the contact wave. Since pressure should be constant across the contact, and, even with the general EOS, a temperature jump needs a density drop for pressure to remain constant, τ should counteract the behavior of T across the contact.

The temperature evolution equation appears as:

$$\frac{\partial T}{\partial t} = -u\frac{\partial T}{\partial x} + \frac{1}{\rho c_p}\left[(1 - \rho h_p)\frac{Dp}{Dt}\right] \tag{5.79}$$

(see, e.g. [5]) where c_p is the specific heat at constant pressure, $c_p = \partial h/\partial T|_p$ and $h_p \equiv \partial h/\partial p|_T$, with $h = e + p/\rho$ the specific enthalpy. We can use the standard

pressure evolution equation (Eq. **??**) to eliminate the Lagrangian pressure term, resulting in:

$$\frac{\partial T}{\partial t} = -u\frac{\partial T}{\partial x} - \eta\frac{\partial u}{\partial x} \tag{5.80}$$

where we defined

$$\eta \equiv \frac{1-\rho h_p}{c_p}c^2 \tag{5.81}$$

The density equation remains unchanged from the traditional primitive variable formulation, but for the velocity, we need to write the $\partial p/\partial x$ term in terms of $\hat{q}$. We do this via the chain rule. We define $p_\rho \equiv \partial p/\partial\rho|_T$ and $p_T \equiv \partial p/\partial T|_\rho$, then our velocity equation is:

$$\frac{\partial u}{\partial t} + u\frac{\partial u}{\partial x} - \frac{p_\rho}{\tau}\frac{\partial \tau}{\partial x} + \tau p_T\frac{\partial T}{\partial x} = 0 \tag{5.82}$$

Note that here we neglected any composition dependence in the EOS.

Our primitive variable system in matrix form is then:

$$\hat{q}_t + \hat{A}\hat{q}_x = 0 \tag{5.83}$$

with

$$\hat{A}(\hat{q}) = \begin{pmatrix} u & -\tau & 0 \\ -\dfrac{p_\rho}{\tau} & u & \tau p_T \\ 0 & \eta & u \end{pmatrix} \tag{5.84}$$

The eigenvalues can be found through the characteristic polynomial, $|\hat{A} - \lambda I| = 0$:

$$(u-\lambda)^3 - (u-\lambda)\left(p_T\eta\tau + p_\rho\right) = 0 \tag{5.85}$$

We can simplify the last term in parenthesis:

$$p_T\eta\tau + p_\rho = \frac{p_T(1-\rho h_p)c^2}{\rho c_p} + p_\rho = \frac{c^2}{c_p}\left(\frac{pp_T}{\rho^2 p_\rho} - \frac{p_T e_\rho}{p_\rho}\right) + p_\rho \tag{5.86}$$

where we substituted in

$$h_p = \frac{1}{\rho}\left(1 - \frac{p}{\rho p_\rho}\right) + \frac{e_\rho}{p_\rho} \tag{5.87}$$

(see [5], appendix A) where $e_\rho = \partial e/\partial\rho|_T$. The term in parenthesis in Eq. 5.86 is simply $c_p - c_v$ ([24], Eq. 9.81) where c_v is the specific heat at constant volume, and Eq. 5.86 further reduces to:

$$p_T\eta\tau + p_\rho = \frac{c^2}{c_p}(c_p - c_v) + p_\rho = c^2 - c^2\frac{\chi_p}{\Gamma_1} + p_\rho = c^2 \tag{5.88}$$

where we used $c_v/c_p = \chi_p/\Gamma_1$ and $\chi_p = \rho p_\rho/p$ ([24], Eqs. 9.87, 9.82). Putting this into our characteristic polynomial, we see that the eigenvalues are, as expected, $\lambda = u, u \pm c$. It is also useful to note that

$$\eta = \frac{c^2 - p_\rho}{\tau p_T} \tag{5.89}$$

We construct the left and right eigenvectors such that they are orthonormal, and find:

$$\hat{R} = \begin{pmatrix} 1 & 1 & 1 \\ c/\tau & 0 & -c/\tau \\ -(c^2 - p_\rho)/\tau^2 p_T & p_\rho/\tau^2 p_T & -(c^2 - p_\rho)/\tau^2 p_T \end{pmatrix} \tag{5.90}$$

and

$$\hat{L} = \begin{pmatrix} p_\rho/2c^2 & \tau/2c & -\tau^2 p_T/2c^2 \\ 1 - p_\rho/c^2 & 0 & \tau^2 p_T/c^2 \\ p_\rho/2c^2 & -\tau/2c & -\tau^2 p_T/2c^2 \end{pmatrix} \tag{5.91}$$

We note that all thermodynamic derivatives are expressed in terms of ρ or T with the other quantity held constant. This is in the form we expect a (T, ρ)-based EOS to return derivatives. Notice also that the temperature jumps across the $(\circ)$ wave (the contact discontinuity; this is seen from the non-zero value in $\hat{r}^{(\circ)}$ for the temperature).

We write:

$$\hat{\beta}_s^{(-)} \equiv (\hat{l}^{(-)} \cdot \Delta\hat{q}^{(-)}) = \frac{1}{2C}\left(\frac{\rho^2 p_\rho}{C}\Delta\tau^{(-)} + \Delta u^{(-)} - \frac{p_T}{C}\Delta T^{(-)}\right) \tag{5.92}$$

$$\hat{\beta}_s^{(\circ)} \equiv (\hat{l}^{(\circ)} \cdot \Delta\hat{q}^{(\circ)}) = \Delta\tau^{(\circ)} + \frac{1}{C^2}\left(-\rho^2 p_\rho \Delta\tau^{(\circ)} + p_T \Delta T^{(\circ)}\right) \tag{5.93}$$

$$\hat{\beta}_s^{(+)} \equiv (\hat{l}^{(+)} \cdot \Delta\hat{q}^{(+)}) = \frac{1}{2C}\left(\frac{\rho^2 p_\rho}{C}\Delta\tau^{(+)} - \Delta u^{(+)} - \frac{p_T}{C}\Delta T^{(+)}\right) \tag{5.94}$$

Note that since $p_\tau = -\rho^2 p_\rho$, we can form a $\Delta p^{(\nu)}$ as

$$\Delta p^{(\nu)} = p_\tau \Delta\tau^{(\nu)} + p_T \Delta T^{(\nu)} \tag{5.95}$$

and then we see that the $\hat{\beta}^{(\nu)}$'s above have the same functional form as the $\mathring{\beta}^{(\nu)}$ from the $\mathring{q} = (\tau, u, p, e)^\intercal$ eigensystem. This is no surprise, since the $l \cdot \Delta q$ are the characteristic variables of the Euler equations. The numerical values will differ though, because the $\hat{\beta}^{(\nu)}$ and $\mathring{\beta}^{(\nu)}$ use different reference states and reconstructed variables.

Finally, we can write out the interface states:

$$\tau_s = \tilde{\tau} - (\hat{\beta}^{(-)} + \hat{\beta}^{(\circ)} + \hat{\beta}^{(+)}) \tag{5.96}$$

$$u_s = \tilde{u} - (C\hat{\beta}^{(-)} - C\hat{\beta}^{(+)}) \tag{5.97}$$

$$T_s = \tilde{T} - \frac{1}{p_T}\left\{\left[-C^2\hat{\beta}^{(-)} - C^2\hat{\beta}^{(+)}\right] + \rho^2 p_\rho\left[\hat{\beta}^{(-)} + \hat{\beta}^{(\circ)} + \hat{\beta}^{(+)}\right]\right\} \tag{5.98}$$

For T_s, we recognize the first quantity in the square brackets as being the same as $-(p_s - \tilde{p})$ in the $\mathring{q}$ system, and the second term in the square brackets being the same as $-(\tau_s - \tilde{\tau})$ in the $\mathring{q}$ system, then we see

$$p_T(T_s - \tilde{T}) \approx (p_s - \tilde{p}) - p_\tau(\tau_s - \tilde{\tau}) \tag{5.99}$$

which is what we would expect for jumps in p when applying the chain rule. Note that this is not a strict equality, since the reference states and interpolation between the two eigensystems are different. Nevertheless, this demonstrates the connection between the two methods.

The above did not consider variations in the composition of the fluid. Multiple species complicate things—now the replacement of the pressure gradient picks up a composition gradient term. The eigensystem will change with this addition. We don't explore this here at this time.

5.6 Multidimensional problems

The multidimensional case is very similar to the multidimensional advection problem. Our system of equations is now:

$$U_t + [F^{(x)}(U)]_x + [F^{(y)}(U)]_y = 0 \tag{5.100}$$

with

$$U = \begin{pmatrix} \rho \\ \rho u \\ \rho v \\ \rho E \end{pmatrix} \qquad F^{(x)}(U) = \begin{pmatrix} \rho u \\ \rho uu + p \\ \rho vu \\ \rho uE + up \end{pmatrix} \qquad F^{(y)}(U) = \begin{pmatrix} \rho v \\ \rho vu \\ \rho vv + p \\ \rho vE + vp \end{pmatrix} \tag{5.101}$$

We note that there is no transformation that can convert the multidimensional system into characteristic variables, since we cannot simultaneously diagonalize the Jacobians corresponding to $F^{(x)}$ and $F^{(y)}$. Related to this is that when limiting, we limit one-dimensional slopes instead of doing a full multidimensional reconstruction and limiting (see [10] for a multidimensional limiting procedure for linear advection. For the Euler equations, since we cannot write the multidimensional system in a characteristic form, we cannot use this type of method).

For a directionally-unsplit discretization, we predict the cell-centered quantities to the edges by Taylor expanding the conservative state, U, in space and time. Now, when replacing the time derivative ($\partial U/\partial t$) with the divergence of the fluxes, we gain a transverse flux derivative term. For example, predicting to the upper x edge

of zone i, j, we have:

$$U^{n+1/2}_{i+1/2,j,L} = U^n_{i,j} + \frac{\Delta x}{2}\frac{\partial U}{\partial x} + \frac{\Delta t}{2}\frac{\partial U}{\partial t} + \dots \tag{5.102}$$

$$= U^n_{i,j} + \frac{\Delta x}{2}\frac{\partial U}{\partial x} - \frac{\Delta t}{2}\frac{\partial F^{(x)}}{\partial x} - \frac{\Delta t}{2}\frac{\partial F^{(y)}}{\partial y} \tag{5.103}$$

$$= U^n_{i,j} + \frac{1}{2}\left[1 - \frac{\Delta t}{\Delta x}A^{(x)}(U)\right]\Delta U - \frac{\Delta t}{2}\frac{\partial F^{(y)}}{\partial y} \tag{5.104}$$

where $A^{(x)}(U) \equiv \partial F^{(x)}/\partial U$. We decompose this into a *normal state* and a *transverse flux difference*. Adopting the notation from Colella (1990), we use $\hat{U}$ to denote the normal state:

$$\hat{U}^{n+1/2}_{i+1/2,j,L} \equiv U^n_{i,j} + \frac{1}{2}\left[1 - \frac{\Delta t}{\Delta x}A^{(x)}(U)\right]\Delta U \tag{5.105}$$

$$U^{n+1/2}_{i+1/2,j,L} = \hat{U}^{n+1/2}_{i+1/2,j,L} - \frac{\Delta t}{2}\frac{\partial F^{(y)}}{\partial y} \tag{5.106}$$

The primitive variable form for this system is

$$q_t + A^{(x)}(q)q_x + A^{(y)}(q)q_y = 0 \tag{5.107}$$

where

$$q = \begin{pmatrix} \rho \\ u \\ v \\ p \end{pmatrix} \qquad A^{(x)}(q) = \begin{pmatrix} u & \rho & 0 & 0 \\ 0 & u & 0 & 1/\rho \\ 0 & 0 & u & 0 \\ 0 & \gamma p & 0 & u \end{pmatrix} \qquad A^{(y)}(q) = \begin{pmatrix} v & 0 & \rho & 0 \\ 0 & v & 0 & 0 \\ 0 & 0 & v & 1/\rho \\ 0 & 0 & \gamma p & v \end{pmatrix} \tag{5.108}$$

There are now 4 eigenvalues. For $A^{(x)}(q)$, they are $u - c$, u, u, $u + c$. If we just look at the system for the x evolution, we see that the transverse velocity (in this case, v) just advects with velocity u, corresponding to the additional eigenvalue.

> **Exercise 5.8:** *Derive the form of $A^{(x)}(q)$ and $A^{(y)}(q)$ and find their left and right eigenvectors.*

We note here that $\hat{U}^{n+1/2}_{i+1/2,j,L}$ is essentially one-dimensional, since only the x-fluxes are involved (through $A^{(x)}(U)$). This means that we can compute this term using the one-dimensional techniques developed in § 5.2. In particular, Colella (1990) suggest that we switch to primitive variables and compute this as:

$$\hat{U}^{n+1/2}_{i+1/2,j,L} = U(\hat{q}^{n+1/2}_{i+1/2,j,L}) \tag{5.109}$$

Similarly, we consider the system projected along the y-direction to define the normal states on the y-edges, again using the one-dimensional reconstruction on the primitive variables from § 5.2:

$$\hat{U}^{n+1/2}_{i,j+1/2,L} = U(\hat{q}^{n+1/2}_{i,j+1/2,L}) \tag{5.110}$$

To compute the full interface state (Eq. 5.106), we need to include the transverse term. Colella (1990) gives two different procedures for evaluating the transverse fluxes. The first is to simply use the cell-centered $U_{i,j}$ (Colella 1990, Eq. 2.13); the second is to use the reconstructed normal states (the $\hat{U}$'s) (Eq. 2.15). In both cases, we need to solve a *transverse Riemann problem* to find the true state on the transverse interface. This latter approach is what we prefer. In particular, for computing the full x-interface left state, $U^{n+1/2}_{i+1/2,j,L}$, we need the transverse (y) states, which we define as

$$U^T_{i,j+1/2} = \mathcal{R}(\hat{U}^{n+1/2}_{i,j+1/2,L}, \hat{U}^{n+1/2}_{i,j+1/2,R}) \tag{5.111}$$

$$U^T_{i,j-1/2} = \mathcal{R}(\hat{U}^{n+1/2}_{i,j-1/2,L}, \hat{U}^{n+1/2}_{i,j-1/2,R}) \tag{5.112}$$

Taken together, the full interface state is now:

$$U^{n+1/2}_{i+1/2,j,L} = U(\hat{q}^{n+1/2}_{i+1/2,j,L}) - \frac{\Delta t}{2}\frac{F^{(y)}(U^T_{i,j+1/2}) - F^{(y)}(U^T_{i,j-1/2})}{\Delta y} \tag{5.113}$$

The right state at the $i + 1/2$ interface can be similarly computed (starting with the data in zone $i + 1, j$ and expanding to the left) as:

$$U^{n+1/2}_{i+1/2,j,R} = U(\hat{q}^{n+1/2}_{i+1/2,j,R}) - \frac{\Delta t}{2}\frac{F^{(y)}(U^T_{i+1,j+1/2}) - F^{(y)}(U^T_{i+1,j-1/2})}{\Delta y} \tag{5.114}$$

Note the indices on the transverse states—they are now to the right of the interface (since we are dealing with the right state).

We then find the x-interface state by solving the Riemann problem normal to our interface:

$$U^{n+1/2}_{i+1/2,j} = \mathcal{R}(U^{n+1/2}_{i+1/2,j,L}, U^{n+1/2}_{i+1/2,j,R}) \tag{5.115}$$

Therefore, construction of the interface states now requires two Riemann solves: a transverse and normal one. The fluxes are then evaluated as:

$$F^{(x),n+1/2}_{i+1/2,j} = F^{(x)}(U^{n+1/2}_{i+1/2,j}) \tag{5.116}$$

Note, for multi-dimensional problems, in the Riemann solver, the transverse velocities are simply selected based on the speed of the contact, giving either the left or right state.

The final conservative update is done as:

$$U^{n+1}_{i,j} = U^n_{i,j} + \frac{\Delta t}{\Delta x}\left(F^{(x),n+1/2}_{i-1/2,j} - F^{(x),n+1/2}_{i+1/2,j}\right) + \frac{\Delta t}{\Delta y}\left(F^{(y),n+1/2}_{i,j-1/2} - F^{(y),n+1/2}_{i,j+1/2}\right) \tag{5.117}$$

5.7 Boundary conditions

Boundary conditions are implemented through ghost cells. The following are the most commonly used boundary conditions. For the expressions below, we use the subscript lo to denote the spatial index of the first valid zone in the domain (just inside the left boundary).

- *Outflow*: the idea here is that the flow should gracefully leave the domain. The simplest form is to simply give all variables a zero-gradient:

$$\begin{pmatrix} \rho_{\mathrm{lo}-1,j} \\ (\rho u)_{\mathrm{lo}-1,j} \\ (\rho v)_{\mathrm{lo}-1,j} \\ (\rho E)_{\mathrm{lo}-1,j} \end{pmatrix} = \begin{pmatrix} \rho_{\mathrm{lo},j} \\ (\rho u)_{\mathrm{lo},j} \\ (\rho v)_{\mathrm{lo},j} \\ (\rho E)_{\mathrm{lo},j} \end{pmatrix} \tag{5.118}$$

  Note that these boundaries are not perfect. At the boundary, one (or more) of the waves from the Riemann problem can still enter the domain. Only for supersonic flow, do all waves point outward.

- *Reflect*: this is appropriate at a solid wall or symmetry plane. All variables are reflected across the boundary, with the normal velocity given the opposite sign. At the x-boundary, the first ghost cell is:

$$\begin{pmatrix} \rho_{\mathrm{lo}-1,j} \\ (\rho u)_{\mathrm{lo}-1,j} \\ (\rho v)_{\mathrm{lo}-1,j} \\ (\rho E)_{\mathrm{lo}-1,j} \end{pmatrix} = \begin{pmatrix} \rho_{\mathrm{lo},j} \\ -(\rho u)_{\mathrm{lo},j} \\ (\rho v)_{\mathrm{lo},j} \\ (\rho E)_{\mathrm{lo},j} \end{pmatrix} \tag{5.119}$$

  The next is:

$$\begin{pmatrix} \rho_{\mathrm{lo}-2,j} \\ (\rho u)_{\mathrm{lo}-2,j} \\ (\rho v)_{\mathrm{lo}-2,j} \\ (\rho E)_{\mathrm{lo}-2,j} \end{pmatrix} = \begin{pmatrix} \rho_{\mathrm{lo}+1,j} \\ -(\rho u)_{\mathrm{lo}+1,j} \\ (\rho v)_{\mathrm{lo}+1,j} \\ (\rho E)_{\mathrm{lo}+1,j} \end{pmatrix} \tag{5.120}$$

  and so on ...

- *Inflow*: inflow boundary conditions specify the state directly on the boundary. Technically, this state is on the boundary itself, not the cell-center. This can be accounted for by modifying the stencils used in the reconstruction near inflow boundaries.

- *Hydrostatic*: a hydrostatic boundary can be used at the base of an atmosphere to provide the pressure support necessary to hold up the atmosphere against gravity while still letting acoustic waves pass through. An example of this is described in [56].

5.8 Higher Order

In the methods above, we predicted a time-centered interface state and used this to evaluate the fluxes through the interface. There are alternate techniques however. For instance, we could do a method of lines approach where we discretize the system in space, leading to a system of ODEs in time, and the we can use a Runge-Kutta integrator to evolve the system in time.

Discretizing our system in space leads to the following system:

$$\frac{dU_{i,j}}{dt} = -\frac{F^{(x)}(U_{i+1/2,j}) - F^{(x)}(U_{i-1/2,j})}{\Delta x} - \frac{F^{(y)}(U_{i,j+1/2}) - F^{(y)}(U_{i,j-1/2})}{\Delta y} \tag{5.121}$$

Note that there is no time superscript in the U used to evaluate the fluxes on the righthand side—we have not done any time discretization yet. Now we can use an ODE integrator to solve this system.

Consider second-order Runge-Kutta. We evaluate two slopes,

$$k_1 = \Delta t \left[-\frac{F^{(x)}(U^n_{i+1/2,j}) - F^{(x)}(U^n_{i-1/2,j})}{\Delta x} - \frac{F^{(y)}(U^n_{i,j+1/2}) - F^{(y)}(U^n_{i,j-1/2})}{\Delta y} \right] \tag{5.122}$$

$$k_2 = \Delta t \left[-\frac{F^{(x)}([U^n + k_1/2]_{i+1/2,j}) - F^{(x)}([U^n + k_1/2]_{i-1/2,j})}{\Delta x} - \frac{F^{(y)}([U^n + k_1/2]_{i,j+1/2}) - F^{(y)}([U^n + k_1/2]_{i,j-1/2})}{\Delta y} \right] \tag{5.123}$$

and then

$$U^{n+1}_{i,j} = U^n_{i,j} + k_2 \tag{5.124}$$

In the construction of the interface states, $U^n_{i+1/2,j}$ or $[U^n + k_1/2]_{i+1/2,j}$, there is no explicit transverse term, since that arose from Taylor expanding $U^n_{i,j}$ in time through $\Delta t/2$. Instead, we simply construct these interface states using a one-dimensional reconstruction and solve a Riemann problem at each interface. The evaluation of the second slope, k_2, implicitly includes the transverse information since we add $k_1/2$ to $U^n_{i,j}$ before doing the prediction to the interfaces. Also note, however, that in this construction of the interface states, there is no characteristic projection, since that arises from predicting the interface states forward in time. Again, these interface states are at a constant time, not predicted into the future.

Generally speaking we want the order of accuracy in time to match that in space. The fourth-order Runge-Kutta method is a popular method for integrating ODEs, so it makes sense to couple this with a fourth-order-in-space method. However, going higher-order than second-order is more challenging. The key issue is that

we can no longer simply approximate the cell average as the cell-center value, i.e., $\langle \phi \rangle_i \neq \phi_i$. This comes into play, for instance, in translating between the conserved and primitive variables. A fully fourth-order method is presented in [36]

An additional complexity arises when doing multiphysics. Often we split the different physical processes up and treat them in turn. There are standard methods to do this with second-order accuracy in time, but higher-order is more tricky.

5.9 Going further

5.9.1 Flattening and Contact Steepening

Shocks are self-steepening (this is how we detect them in the Riemann solver—we look for converging characteristics). This can cause trouble with the methods here, because the shocks may become too steep.

Flattening is a procedure to add additional dissipation at shocks, to ensure that they are smeared out over ~ 2 zones. The flattening procedure is a multi-dimensional operation that looks at the pressure and velocity profiles and returns a coefficient, $\chi \in [0, 1]$ that multiplies the limited slopes. The convention most sources use is that $\chi = 1$ means no flattening (the slopes are unaltered), while $\chi = 0$ means complete flattening—the slopes are zeroed, dropping us to a first-order method. See for example in Saltzman [45]. Once the flattening coefficient is determined, the interface state is blended with the cell-centered value via:

$$q^{n+1/2}_{i+1/2,\{L,R\}} \leftarrow (1 - \chi) q_i + \chi q^{n+1/2}_{i+1/2,\{L,R\}} \tag{5.125}$$

Note that the flattening algorithm increases the stencil size of piecewise-linear and piecewise-parabolic reconstruction to 4 ghost cells on each side. This is because the flattening procedure itself looks at the pressure 2 zones away, and we need to construct the flattening coefficient in both the first ghost cell (since we need the interface values there) and the second ghost cell (since the flattening procedure looks at the coefficients in its immediate upwinded neighbor).

In contrast to shocks, contact waves do not steepen (they are associated with the middle characteristic wave, and the velocity does not change across that, meaning there cannot be any convergence). The original PPM paper advocates a contact steepening method to artificially steepen contact waves. While it shows good results in 1-d, it can be problematic in multi-dimensions.

Overall, the community seems split over whether this term should be used. Many people advocate that if you reach a situation where you think contact steepening may be necessary, it is more likely that the issue is that you do not have enough resolution.

5.9.2 Artificial viscosity

Colella and Woodward argue discuss that behind slow-moving shocks these methods can have oscillations. The fix they propose is to use some artificial viscosity—this is additional dissipation that kicks in at shocks. (They argue that flattening alone is not enough).

We use a multidimensional analog of their artificial viscosity ([23], Eq. 4.5) which modifies the fluxes. By design, it only kicks in for converging flows, such that you would find around a shock.

5.9.3 Species

For multifluid flows, the Euler equations are augments with continuity equations for each of the (chemical or nuclear) species:

$$\frac{\partial(\rho X_k)}{\partial t} + \frac{\partial(\rho X_k u)}{\partial x} = 0 \tag{5.126}$$

here, X_k are the mass fractions and obey $\sum_k X_k = 1$. Using the continuity equation, we can write this as an advection equation:

$$\frac{\partial X_k}{\partial t} + u\frac{\partial X_k}{\partial x} = 0 \tag{5.127}$$

When we now consider our primitive variables: $q = (\rho, u, p, X_k)$, we find

$$A(q) = \begin{pmatrix} u & \rho & 0 & 0 \\ 0 & u & 1/\rho & 0 \\ 0 & \gamma p & u & 0 \\ 0 & 0 & 0 & u \end{pmatrix} \tag{5.128}$$

There are now 4 eigenvalues, with the new one also being simply u. This says that the species simply advect with the flow. The right eigenvectors are now:

$$r^{(1)} = \begin{pmatrix} 1 \\ -c/\rho \\ c^2 \\ 0 \end{pmatrix} \qquad r^{(2)} = \begin{pmatrix} 1 \\ 0 \\ 0 \\ 0 \end{pmatrix} \qquad r^{(3)} = \begin{pmatrix} 0 \\ 0 \\ 0 \\ 1 \end{pmatrix} \qquad r^{(4)} = \begin{pmatrix} 1 \\ c/\rho \\ c^2 \\ 0 \end{pmatrix} \tag{5.129}$$

corresponding to $\lambda^{(1)} = u - c$, $\lambda^{(2)} = u$, $\lambda^{(3)} = u$, and $\lambda^{(4)} = u + c$. We see that for the species, the only non-zero element is for one of the u eigenvectors. This means that X_k only jumps over this middle wave. In the Riemann solver then, there is no 'star' state for the species, it just jumps across the contact wave.

To add species into the solver, you simply need to reconstruct X_k as described above, find the interface values using this new $A(q)$ and associated eigenvectors, solve the Riemann problem, with X_k on the interface being simply the left or right

state depending on the sign of the contact wave speed, and do the conservative update for ρX_k using the species flux.

One issue that can arise with species is that even if $\sum_k X_k = 1$ initially, after the update, that may no longer be true. There are a variety of ways to handle this:

- You can update the species, (ρX_k) to the new time and then define the density to be $\rho = \sum_k (\rho X_k)$—this means that you are not relying on the value of the density from the mass continuity equation itself.
- You can force the interface states of X_k to sum to 1. Because the limiting is non-linear, this is where problems can arise. If the interface values of X_k are forced to sum to 1 (by renormalizing), then the updated cell-centered value of X_k will as well. This is the approach discussed in [43].
- You can design the limiting procedure to preserve the summation property. This approach is sometimes taken in the combustion field. For piecewise linear reconstruction, this can be obtained by computing the limited slopes of all the species, and taking the most restrictive slope and applying this same slope to all the species.

5.9.4 Source terms

Adding source terms is straightforward. For a system described by

$$U_t + [F^{(x)}(U)]_x + [F^{(y)}(U)]_y = H \tag{5.130}$$

we predict to the edges in the same fashion as described above, but now when we replace $\partial U/\partial t$ with the divergence of the fluxes, we also pick up the source term. This appears as:

$$U^{n+1/2}_{i+1/2,j,L} = U^n_{i,j} + \frac{\Delta x}{2}\frac{\partial U}{\partial x} + \frac{\Delta t}{2}\frac{\partial U}{\partial t} + \ldots \tag{5.131}$$

$$= U^n_{i,j} + \frac{\Delta x}{2}\frac{\partial U}{\partial x} - \frac{\Delta t}{2}\frac{\partial F^{(x)}}{\partial x} - \frac{\Delta t}{2}\frac{\partial F^{(y)}}{\partial y} + \frac{\Delta t}{2}H_{i,j} \tag{5.132}$$

$$= U^n_{i,j} + \frac{1}{2}\left[1 - \frac{\Delta t}{\Delta x}A^{(x)}(U)\right]\Delta U - \frac{\Delta t}{2}\frac{\partial F^{(y)}}{\partial y} + \frac{\Delta t}{2}H_{i,j} \tag{5.133}$$

We can compute things as above, but simply add the source term to the $\hat{U}$'s and carry it through.

Note that the source here is cell-centered. This expansion is second-order accurate. This is the approach outlined in Miller & Colella [37].

The original PPM paper does things differently. They construct a parabolic profile of g in each zone and integrate under that profile to determine the average g carried by each wave to the interface. Finally, they include the gravitational source

term in the characteristic projection itself. To see this, write our system in primitive form as:

$$q_t + A(q)q_x = G \tag{5.134}$$

where $G = (0, g, 0)^T$—i.e. the gravitational source only affects u, not ρ or p. Note that in the PPM paper, they put G on the lefthand side of the primitive variable equation, so our signs are opposite. Our projections are now:

$$\sum_{\nu;\lambda^{(\nu)}\geq 0} l^{(\nu)} \cdot (\tilde{q} - \mathcal{I}_+^{(\nu)}(q) - \tfrac{\Delta t}{2} G) r^{(\nu)} \tag{5.135}$$

for the left state, and

$$\sum_{\nu;\lambda^{(\nu)}\leq 0} l^{(\nu)} \cdot (\tilde{q} - \mathcal{I}_-^{(\nu)}(q) - \tfrac{\Delta t}{2} G) r^{(\nu)} \tag{5.136}$$

for the right state. Since G is only non-zero for velocity, only the velocity changes. Writing out the sum (and performing the vector products), we get:

$$\begin{aligned} u_{i+1/2,L}^{n+1/2} = \tilde{u}_+ &- \frac{1}{2}\left[\left(\tilde{u}_+ - \mathcal{I}_+^{(-)}(u) - \frac{\Delta t}{2}\mathcal{I}_+^{(-)}(g)\right) - \frac{\tilde{p}_+ - \mathcal{I}_+^{(-)}(p)}{C}\right] \\ &- \frac{1}{2}\left[\left(\tilde{u}_+ - \mathcal{I}_+^{(+)}(u) - \frac{\Delta t}{2}\mathcal{I}_+^{(+)}(g)\right) + \frac{\tilde{p}_+ - \mathcal{I}_+^{(+)}(p)}{C}\right] \end{aligned} \tag{5.137}$$

where the only change from Eq. 5.62 are the $\mathcal{I}_+^{(-)}(g)$ and $\mathcal{I}_+^{(+)}(g)$ terms.

These differ from the expression in the PPM paper, where $\Delta t G$, not $\Delta t/2G$ is used in the projection, however this appears to be a typo. To see this, notice that if both waves are moving toward the interface, then the source term that is added to the interface state is $(\Delta t/4)(\mathcal{I}_+^{(-)}(g) + \mathcal{I}_+^{(+)}(g))$ for the left state, which reduces to $(\Delta t/2)g$ for constant g—this matches the result from the Taylor expansion above (Eq. 5.133).

5.9.5 General equation of state

The above methods were formulated with a constant gamma equation of state. A general equation of state (such as degenerate electrons) requires a more complex method. The classic prescription for extending this methodology is presented by Colella and Glaz [20]. They construct a thermodynamic index,

$$\gamma_e = \frac{p}{\rho e} + 1 \tag{5.138}$$

and derive an evolution equation for γ_e (C&G, Eq. 26). We can derive a similar expression as

$$\begin{aligned} \frac{D\gamma_e}{Dt} &= \frac{D}{Dt}\left(\frac{p}{\rho e} + 1\right) = -\frac{p}{(\rho e)^2}\frac{D(\rho e)}{Dt} + \frac{1}{\rho e}\frac{Dp}{Dt} \\ &= (\gamma_e - 1)(\gamma_e - \Gamma_1)\nabla \cdot U \end{aligned} \tag{5.139}$$

where we used Eqs. 5.14 and 5.78, and the definition of the sound speed.

This evolution equation is used to predict γ_e to interfaces, and these interface values of γ_e are used in the Riemann solver presented there to find the fluxes through the interface. A different adiabatic index (they call Γ, we call Γ_1) appears in the definition of the sound speed. They argue that this can be brought to interfaces in a piecewise constant fashion while still making the overall method second order, since Γ_1 does not explicitly appear in the fluxes (see the discussion at the top of page 277).

Alternately, the Castro paper [2] relies on an idea from an unpublished manuscript by Colella, Glaz, and Ferguson that predicts ρe to edges in addition to ρ, u, and p. Since ρe comes from a conservation-like equation (Eq. 5.78, predicting it to the interface in the unsplit formulation is straightforward. This over-specifies the thermodynamics, but eliminates the need for γ_e.

With the addition of ρe, our system becomes:

$$q = \begin{pmatrix} \rho \\ u \\ p \\ \rho e \end{pmatrix} \qquad A = \begin{pmatrix} u & \rho & 0 & 0 \\ 0 & u & 1/\rho & 0 \\ 0 & \rho c^2 & u & 0 \\ 0 & \rho h & 0 & u \end{pmatrix} \tag{5.140}$$

where $h = e + p/\rho$ is the specific enthalpy. The eigenvalues of this system are:

$$\lambda^{(1)} = u - c \qquad \lambda^{(2)} = u \qquad \lambda^{(3)} = u \qquad \lambda^{(4)} = u + c \tag{5.141}$$

and the eigenvectors are:

$$r^{(1)} = \begin{pmatrix} 1 \\ -c/\rho \\ c^2 \\ h \end{pmatrix} \qquad r^{(2)} = \begin{pmatrix} 1 \\ 0 \\ 0 \\ 0 \end{pmatrix} \qquad r^{(3)} = \begin{pmatrix} 0 \\ 0 \\ 0 \\ 1 \end{pmatrix} \qquad r^{(4)} = \begin{pmatrix} 1 \\ c/\rho \\ c^2 \\ h \end{pmatrix} \tag{5.142}$$

and

$$\begin{aligned} l^{(1)} &= \begin{pmatrix} 0 & -\frac{\rho}{2c} & \frac{1}{2c^2} & 0 \end{pmatrix} \\ l^{(2)} &= \begin{pmatrix} 1 & 0 & -\frac{1}{c^2} & 0 \end{pmatrix} \\ l^{(3)} &= \begin{pmatrix} 0 & 0 & -\frac{h}{c^2} & 1 \end{pmatrix} \\ l^{(4)} &= \begin{pmatrix} 0 & \frac{\rho}{2c} & \frac{1}{2c^2} & 0 \end{pmatrix} \end{aligned} \tag{5.143}$$

Remember that the state variables in the q vector are mixed into the other states by $l \cdot q$. Since all $l^{(\nu)}$'s have 0 in the ρe 'slot' (the last position) except for $l^{(3)}$, and the corresponding $r^{(3)}$ is only non-zero in the ρe slot, this means that ρe is not mixed into the other state variables. This is as expected, since ρe is not needed in the system.

Also recall that the jump carried by the wave ν is proportional to $r^{(\nu)}$—since $r^{(1)}$, $r^{(3)}$, and $r^{(4)}$ have non-zero ρe elements, this means that ρe jumps across these three waves.

Working through the sum for the (ρe) state, and using a $\sim$ to denote the reference states, we arrive at:

$$\begin{aligned}(\rho e)^{n+1/2}_{i+1/2,L} = \widetilde{(\rho e)} &- \frac{1}{2}\left[-\frac{\rho}{c}\left(\tilde{u} - \mathcal{I}^{(1)}_{+}(u)\right) + \frac{1}{c^2}\left(\tilde{p} - \mathcal{I}^{(1)}_{+}(p)\right)\right] h \\ &- \left[-\frac{h}{c^2}\left(\tilde{p} - \mathcal{I}^{(3)}_{+}(p)\right) + \left(\widetilde{(\rho e)} - \mathcal{I}^{(3)}_{+}(\rho e)\right)\right] \\ &- \frac{1}{2}\left[\frac{\rho}{c}\left(\tilde{u} - \mathcal{I}^{(4)}_{+}(u)\right) + \frac{1}{c^2}\left(\tilde{p} - \mathcal{I}^{(4)}_{+}(p)\right)\right] h \end{aligned} \tag{5.144}$$

This is the expression that is found in the Castro code.

All of these methods are designed to avoid EOS calls where possible, since general equations of state can be expensive.

Extending these to an unsplit formulation requires carrying an additional auxiliary variable from the primitive state back to the conserved state and adding the transverse gradients to its interface state. Castro deals with a conserved state of $U = (\rho, \rho U, \rho E, p)$, and explicitly adds the transverse terms found in the multidimensional form of Eq. 5.14 to the normal states of p.

5.9.6 Axisymmetry

It is common to so 2-d axisymmetric models—here the r and z coordinates from a cylindrical geometry are modeled. This appears Cartesian, except there is a volume factor implicit in the divergence that must be accounted for. Our system in cylindrical coordinates is:

$$\frac{\partial U}{\partial t} + \frac{1}{r}\frac{\partial r F^{(r)}}{\partial r} + \frac{\partial F^{(z)}}{\partial z} = 0 \tag{5.145}$$

Expanding out the r derivative, we can write this as:

$$\frac{\partial U}{\partial t} + \frac{\partial F^{(r)}}{\partial r} + \frac{\partial F^{(z)}}{\partial z} = -\frac{F^{(r)}}{r} \tag{5.146}$$

This latter form is used when predicting the interface states, with the volume source that appears on the right treated as a source term to the interface states (as described above). Once the fluxes are computed, the final update uses the conservative form of the system, with the volume factors appearing now in the definition of the divergence.

5.9.7 Defining temperature

Although not needed for the pure Euler equations, it is sometimes desirable to define the temperature for source terms (like reactions) or complex equations of state. The temperature can typically be found from the equation of state given the internal energy:

$$e = E - \frac{1}{2}u^2 \tag{5.147}$$

$$T = T(e, \rho) \tag{5.148}$$

Trouble can arise when you are in a region of flow where the kinetic energy dominates (high Mach number flow). In this case, the e defined via subtraction can become negative due to truncation error in the evolution of u compared to E. In this instance, one must either impose a floor value for e or find an alternate method of deriving it.

In [15], an alternate formulation of the Euler equations is proposed. Both the total energy equation *and* the internal energy equation are evolved in each zone. When the flow is dominated by kinetic energy, then the internal energy from the internal energy evolution equation is used. The cost of this is conservation—the internal energy is not a conserved quantity, and switching to it introduces conservation of energy errors.

5.9.8 Limiting on characteristic variables

Some authors (see for example, [48] Eqs. 37, 38) advocate limiting on the characteristic variables rather than the primitive variables. The characteristic slopes for the quantity carried by the wave ν can be found from the primitive variables as:

$$\Delta w^{(\nu)} = l^{(\nu)} \cdot \Delta q \tag{5.149}$$

any limiting would then be done to $\Delta w^{(\nu)}$ and the limited primitive variables would be recovered as:

$$\overline{\Delta q} = \sum_\nu \overline{\Delta w}^{(\nu)} r^{(\nu)} \tag{5.150}$$

(here we use an overline to indicate limiting).

This is attractive because it is more in the spirit of the linear advection equation and the formalism that was developed there. A potential downside is that when you limit on the characteristic variables and convert back to the primitive, the primitive variables may now fall outside of valid physical ranges (for example, negative density).

5.9.9 3-d unsplit

The extension of the unsplit methodology to 3-d is described by Saltzman [45]. The basic idea is the same as in 2-d, except now additional transverse Riemann solve are needed to fully couple in the corners.

Chapter 6

Elliptic Equations and Multigrid

These summarize multigrid on cell-centered grids. The text "A Multigrid Tutorial" [14] is an incredible reference. These notes discuss the basics and point out some specific details for cell-centered grids.

6.1 Elliptic equations

The simplest elliptic PDE is *Laplace's equation*:

$$\nabla^2 \phi = 0 \tag{6.1}$$

Only slightly more complex is *Poisson's equation* (Laplace + a source term):

$$\nabla^2 \phi = f \tag{6.2}$$

These equations can arise in electrostatics (for the electric potential), solving for the gravitational potential from a mass distribution, or enforcing a divergence constraint on a vector field (we'll see this when we consider incompressible flow).

Another common elliptic equation is the *Helmholtz equation*:

$$(\alpha - \nabla \cdot \beta \nabla)\phi = f \tag{6.3}$$

A Helmholtz equation can arise, for example, from a time-dependent equation (like diffusion) by discretizing in time.

Notice that there is no time-dependence in any of these equations. The quantity ϕ is specified instantaneously in the domain subject to boundary conditions.

6.2 Fourier Method

A direct way of solving a constant-coefficient elliptic equation is using Fourier transforms. Using a general Fourier transform (which we consider here) works only for periodic boundary conditions, but other basis functions can be used for other boundary conditions.

Consider the Poisson equation:

$$\nabla^2\phi = f \tag{6.4}$$

We will difference this in a second-order accurate fashion. Thinking of the Laplacian as $\nabla^2\phi = \nabla \cdot \nabla\phi$, we first compute the gradient of ϕ on edges:

$$[\nabla\phi \cdot \hat{x}]_{i+1/2,j} = \frac{\phi_{i+1,j} - \phi_{i,j}}{\Delta x} \tag{6.5}$$

Since this is defined on edges, this represents a centered difference, and is therefore second-order accurate. We then difference the edge-centered gradients to the center to get the Laplacian at cell-centers:

$$\begin{aligned}[\nabla^2\phi]_{i,j} &= \frac{[\nabla\phi \cdot \hat{x}]_{i+1/2,j} - [\nabla\phi \cdot \hat{x}]_{i-1/2,j}}{\Delta x} + \frac{[\nabla\phi \cdot \hat{y}]_{i,j+1/2} - [\nabla\phi \cdot \hat{y}]_{i,j-1/2}}{\Delta y} \\ &= \frac{\phi_{i+1,j} - 2\phi_{i,j} + \phi_{i-1,j}}{\Delta x^2} + \frac{\phi_{i,j+1} - 2\phi_{i,j} + \phi_{i,j-1}}{\Delta y^2} = f_{i,j}\end{aligned} \tag{6.6}$$

Again, since we used a centered-difference of the edge values, this expression is second-order accurate. This is the standard *5-point stencil* for the 2-d Laplacian.

We now assume that we have an FFT subroutine that can take our discrete real-space data, $\phi_{i,j}$ and return the discrete Fourier coefficients, Φ_{k_x,k_y}, and likewise for the source term:

$$\Phi_{k_x,k_y} = \mathcal{F}(\phi_{i,j}) \quad F_{k_x,k_y} = \mathcal{F}(f_{i,j}) \tag{6.7}$$

The power of the Fourier method is that derivatives in real space are multiplications in Fourier space, which makes the solution process in Fourier space straightforward.

We now express $\phi_{i,j}$ and $f_{i,j}$ as sums over their Fourier components. Note, because we are using i as the grid index, we will use i as the imaginary unit:

$$\phi_{i,j} = \frac{1}{MN}\sum_{k_x=0}^{M-1}\sum_{k_y=0}^{N-1}\Phi_{k_x,k_y}e^{2\pi\mathrm{i}ik_x/M}e^{2\pi\mathrm{i}jk_y/N} \tag{6.8}$$

$$f_{i,j} = \frac{1}{MN}\sum_{k_x=0}^{M-1}\sum_{k_y=0}^{N-1}F_{k_x,k_y}e^{2\pi\mathrm{i}ik_x/M}e^{2\pi\mathrm{i}jk_y/N} \tag{6.9}$$

Inserting these into the differenced equation (and dropping the sums and normalization, to focus on a single mode), we have:

$$\frac{1}{MN}\sum_{k_x=0}^{M-1}\sum_{k_y=0}^{N-1}\left\{\frac{\Phi_{k_x,k_y}}{\Delta x^2}e^{2\pi \mathrm{i} jk_y/N}\left[e^{2\pi \mathrm{i}(i+1)k_x/M}-2e^{2\pi \mathrm{i} ik_x/M}+e^{2\pi \mathrm{i}(i-1)k_x/M}\right]+\right.$$
$$\left.\frac{\Phi_{k_x,k_y}}{\Delta y^2}e^{2\pi \mathrm{i} ik_x/M}\left[e^{2\pi \mathrm{i}(j+1)k_y/N}-2e^{2\pi \mathrm{i} jk_y/N}+e^{2\pi \mathrm{i}(j-1)k_y/N}\right]\right\}=$$
$$\frac{1}{MN}\sum_{k_x=0}^{M-1}\sum_{k_y=0}^{N-1}F_{k_x,k_y}e^{2\pi \mathrm{i} ik_x/M}e^{2\pi \mathrm{i} jk_y/N} \tag{6.10}$$

We can bring the righthand side into the sums on the left, and we can then look at just a single (k_x, k_y) term in the series:

$$e^{2\pi \mathrm{i} ik_x/M}e^{2\pi \mathrm{i} jk_y/N}\left\{\frac{\Phi_{k_x,k_x}}{\Delta x^2}\left[e^{2\pi \mathrm{i} k_x/M}+e^{-2\pi \mathrm{i} k_x/M}-2\right]+\right.$$
$$\left.\frac{\Phi_{k_x,k_x}}{\Delta y^2}\left[e^{2\pi \mathrm{i} k_y/N}+e^{-2\pi \mathrm{i} k_y/N}-2\right]-F_{k_x,k_y}\right\}=0 \tag{6.11}$$

Simplifying, we have:

$$\Phi_{k_x,k_y}=\frac{1}{2}\frac{F_{k_x,k_y}}{\left[\cos(2\pi k_x/M)-1\right]\Delta x^{-2}+\left[\cos(2\pi k_y/N)-1\right]\Delta y^{-2}} \tag{6.12}$$

This is the algebraic solution to the Poisson equation in Fourier (frequency) space. Once we evaluate this, we can get the real-space solution by doing the inverse transform:

$$\phi_{i,j}=\mathcal{F}^{-1}(\Phi_{k_x,k_y}) \tag{6.13}$$

We test this technique with the source term:

$$f=8\pi^2\cos(4\pi y)\left[\cos(4\pi x)-\sin(4\pi x)\right]-$$
$$16\pi^2\left[\sin(4\pi x)\cos(2\pi y)^2+\sin(2\pi x)^2\cos(4\pi y)\right] \tag{6.14}$$

which has the analytic solution[1]:

$$\phi=\sin(2\pi x)^2\cos(4\pi y)+\sin(4\pi x)\cos(2\pi y)^2 \tag{6.15}$$

Figure 6.1 shows the solution.

The main downside of this approach is that, because we solve for a single component independently (Eq. 6.12), this only works for linear problems with constant coefficients. This makes it an excellent choice for cosmological problems solving

[1]Note: throughout this chapter, we devise test problems by picking a function that meets the desired boundary conditions and then inserting it into the analytic equation we are solving to find the righthand side

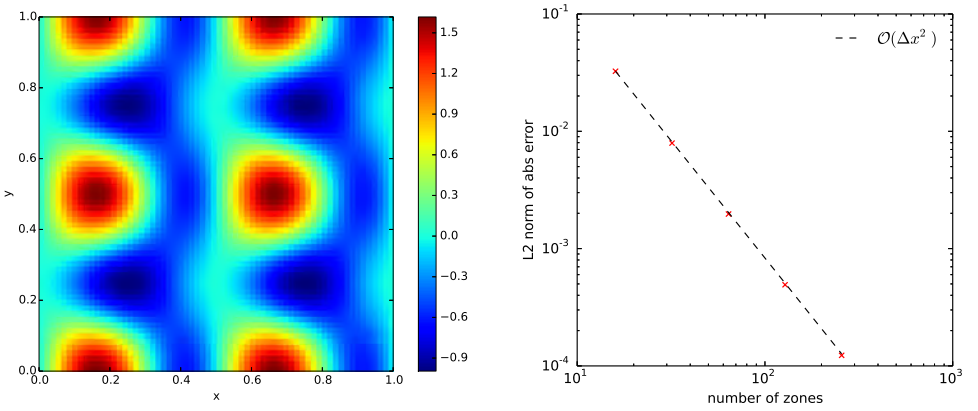

Figure 6.1: (left) Solution to the Poisson equation on a 64^2 grid with source from Eq. 6.14. (right) Error vs. the true solution as a function of resolution for the Fourier method, showing second-order convergence. hydro_examples: `poisson_fft.py`

the gravitational Poisson equation with periodic boundaries on all sides of the domain. However, for a problem like:

$$\nabla \cdot (\beta \nabla \phi) = f \tag{6.16}$$

there would be "cross-talk" between the Fourier modes of β and ϕ, and we would not be able to solve for a single mode of Φ_{k_x,k_y} independently. We discuss methods for these forms next.

6.3 Relaxation

Relaxation is an iterative technique, and as we will see shortly, it provides the basis for the multigrid technique.

Consider Poisson's equation differenced as:

$$\frac{\phi_{i+1,j} - 2\phi_{i,j} + \phi_{i-1,j}}{\Delta x^2} + \frac{\phi_{i,j+1} - 2\phi_{i,j} + \phi_{i,j-1}}{\Delta y^2} = f_{i,j} \tag{6.17}$$

This is a 5-point stencil: for each zone (i, j), we couple in the zones ± 1 in x and ± 1 in y. This discretization uses the standard form for the second derivative, and is second-order accurate in space.

For the moment, consider the case where $\Delta x = \Delta y$. If we solve this discretized equation for $\phi_{i,j}$, then we have:

$$\phi_{i,j} = \frac{1}{4}(\phi_{i+1,j} + \phi_{i-1,j} + \phi_{i,j+1} + \phi_{i,j-1} - \Delta x^2 f_{i,j}) \tag{6.18}$$

A similar expression exists for every zone in our domain, coupling all the zones together. We can't separate the solution of $\phi_{i,j}$ for the neighboring zones, but instead

can apply an iterative technique called *relaxation* (also sometimes called *smoothing* because generally speaking the solution to elliptic equations is a smooth function) to find the solution for ϕ everywhere. Imagine an initial guess to ϕ: $\phi_{i,j}^{(0)}$. We can improve that guess by using our difference equation to define a new value of ϕ, $\phi_{i,j}^{(1)}$:

$$\phi_{i,j}^{(1)} = \frac{1}{4}(\phi_{i+1,j}^{(0)} + \phi_{i-1,j}^{(0)} + \phi_{i,j+1}^{(0)} + \phi_{i,j-1}^{(0)} - \Delta x^2 f_{i,j}) \tag{6.19}$$

or generally, the $k+1$ iteration will see:

$$\phi_{i,j}^{(k+1)} = \frac{1}{4}(\phi_{i+1,j}^{(k)} + \phi_{i-1,j}^{(k)} + \phi_{i,j+1}^{(k)} + \phi_{i,j-1}^{(k)} - \Delta x^2 f_{i,j}) \tag{6.20}$$

This will (slowly) converge to the true solution, since each zone is coupled to each other zone (and to the boundary values that we need to specify—more on that in a moment). This form of relaxation is called *Jacobi iteration*. To implement this, you need two copies of ϕ—the old iteration value and the new iteration value.

An alternate way to do the relaxation is to update $\phi_{i,j}$ in place, as soon as the new value is known. Thus the neighboring cells will see a mix of the old and new solutions. We can express this in-place updating as:

$$\phi_{i,j} \leftarrow \frac{1}{4}(\phi_{i+1,j} + \phi_{i-1,j} + \phi_{i,j+1} + \phi_{i,j-1} - \Delta x^2 f_{i,j}) \tag{6.21}$$

This only requires a single copy of ϕ to be stored. This technique is called *Gauss-Seidel iteration*. A host of other relaxation methods exist, including linear combinations of these two. The text by Briggs is an excellent reference for this, and discusses the strengths of these different approaches.

Next consider the Helmholz equation with constant coefficients:

$$(\alpha - \beta\nabla^2)\phi = f \tag{6.22}$$

We can discretize this as:

$$\alpha\phi_{i,j} - \beta\left(\frac{\phi_{i+1,j} - 2\phi_{i,j} + \phi_{i-1,j}}{\Delta x^2} + \frac{\phi_{i,j+1} - 2\phi_{i,j} + \phi_{i,j-1}}{\Delta y^2}\right) = f_{i,j} \tag{6.23}$$

and the update of $\phi_{i,j}$ through relaxation is:

$$\phi_{i,j} \leftarrow \left(f_{i,j} + \frac{\beta}{\Delta x^2}\phi_{i+1,j} + \frac{\beta}{\Delta x^2}\phi_{i-1,j} + \frac{\beta}{\Delta y^2}\phi_{i,j+1} + \frac{\beta}{\Delta y^2}\phi_{i,j-1}\right) \Big/ \left(\alpha + \frac{2\beta}{\Delta x^2} + \frac{2\beta}{\Delta y^2}\right) \tag{6.24}$$

Notice that if $\alpha = 0$, $\beta = -1$, and $\Delta x = \Delta y$, we recover the relaxation expression for Poisson's equation from above.

6.3.1 Boundary conditions

When using a cell-centered grid, no points fall exactly on the boundary, so we need to use ghost cells to specify boundary conditions. A single ghost cell is sufficient. The common types of boundary conditions are *Dirichlet* (specified value on the boundary), *Neumann* (specified first derivative on the boundary), and periodic. Some restrictions apply. For example, consider $\phi_{xx} = 0$. The solution to this is a line, $\phi = ax + b$. We can specify different Neumann boundary conditions on each end, $\phi_x|_{x=x_l} = p$, $\phi_x|_{x=x_r} = q$, because this specifies two incompatible slopes for our line.

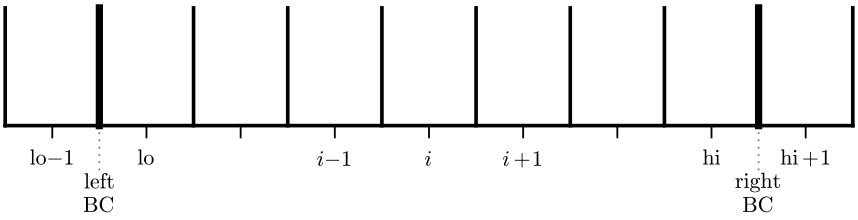

Figure 6.2: The cell-centered grid showing the cells (and ghost cells) surrounding the boundaries and indicating that the boundary conditions are actually specified right at the boundary itself.

Consider Dirichlet boundary conditions, specifying values ϕ_l on the left and ϕ_r on the right boundaries.[2] To second order, we can define these via:

$$\phi_l = \frac{1}{2}(\phi_{\rm lo} + \phi_{\rm lo-1}) \tag{6.25}$$

$$\phi_r = \frac{1}{2}(\phi_{\rm hi} + \phi_{\rm hi+1}) \tag{6.26}$$

This then tells us that the values we need to assign to the ghost cells are:

$$\phi_{\rm lo-1} = 2\phi_l - \phi_{\rm lo} \tag{6.27}$$

$$\phi_{\rm hi+1} = 2\phi_r - \phi_{\rm hi} \tag{6.28}$$

If we instead consider Neumann boundary conditions, we specify values of the derivative on the boundaries: $\phi_x|_l$ on the left and $\phi_x|_r$ on the right. We note that a single difference across the boundary is second-order accurate on the boundary (it is a centered-difference there), so to second-order:

$$\phi_x|_l = \frac{\phi_{\rm lo} - \phi_{\rm lo-1}}{\Delta x} \tag{6.29}$$

$$\phi_x|_r = \frac{\phi_{\rm hi+1} - \phi_{\rm hi}}{\Delta x} \tag{6.30}$$

[2]If the value, ϕ_l or ϕ_r is zero, we call this a *homogeneous boundary condition*. Otherwise we call it an *inhomogeneous boundary condition*

This then tells us that the ghost cells are filled as:

$$\phi_{\mathrm{lo}-1} = \phi_{\mathrm{lo}} - \Delta x\, \phi_x|_l \tag{6.31}$$
$$\phi_{\mathrm{hi}+1} = \phi_{\mathrm{hi}} + \Delta x\, \phi_x|_r \tag{6.32}$$

6.3.2 Residual and true error

The *residual error* is a measure of how well our discrete solution satisfies the discretized equation. For the Poisson equation, we can the residual as:

$$r_{i,j} = f_{i,j} - (L\phi)_{i,j} \tag{6.33}$$

and the residual error as:

$$\epsilon^{(r)} = \|r\| \tag{6.34}$$

where L represents our discretized Laplacian. Note that r is the error with respect to the discrete form of the equation. The true error is the measure of how well our discrete solution approximates the true solution. If ϕ^{true} satisfies $\nabla^2\phi^{\mathrm{true}} = f$, then the true error in each zone is

$$e_{i,j} = \phi^{\mathrm{true}}(x_i, y_j) - \phi_{i,j} \tag{6.35}$$

and

$$\epsilon^{\mathrm{true}} = \|e_{i,j}\| \tag{6.36}$$

We can make $\epsilon^{(r)}$ approach machine precision by performing more and more relaxation iterations, but after some point, this will no longer improve ϵ. The only way to improve ϵ^{true} is to make Δx and Δy smaller. In practice we do not know the true solution so we cannot compute ϵ^{true} and will instead have to rely on $\epsilon^{(r)}$ to monitor our error.

Note that since our operator is linear,

$$Le = L\phi^{\mathrm{true}} - L\phi = f - L\phi = r \tag{6.37}$$

so the error in our solution obeys a Poisson equation with the residual as the source.

6.3.3 Performance

Consider the simple Poisson problem on $x \in [0, 1]$:

$$\phi_{xx} = \sin(x), \qquad \phi(0) = \phi(1) = 0 \tag{6.38}$$

The analytic solution to this is simply

$$\phi^a(x) = -\sin(x) + x\sin(1) \tag{6.39}$$

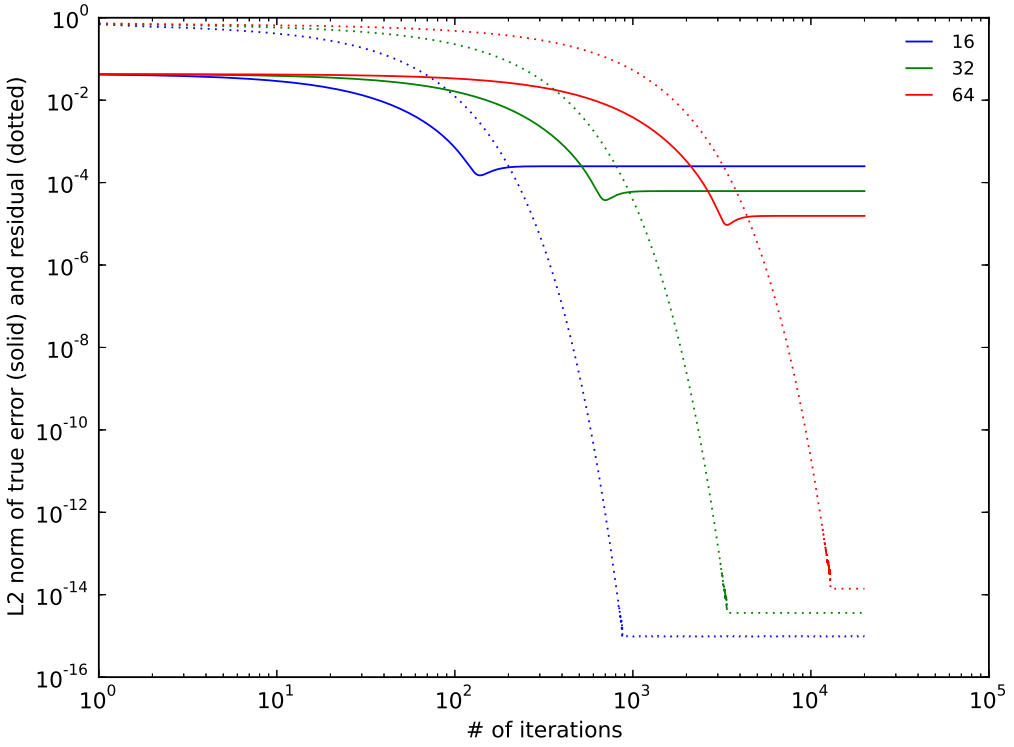

Figure 6.3: Gauss-Seidel relaxation applied to $\phi_{xx} = \sin(x)$ with $\phi(0) = \phi(1) = 0$. Shown are the L2 norm of the error compared with the true solution (solid lines) and the L2 norm of the residual (dotted lines) for 3 different resolutions (16, 32, and 64 zones).

We can perform smoothing and compute both the error against the analytic solution (the 'true' error), $e \equiv \|\phi^a(x_i) - \phi_i\|_2$ and the residual error, $\|r_i\|_2$. Figure 6.3 shows these errors as a function of the number of smoothing iterations for 3 different resolutions.

Notice that the true error stalls at a relatively high value—this is the truncation error of the method. From one resolution to the next, the true error changes as Δx^2, indicating that we are converging as our method should. No additional amount of smoothing will change this—we are getting the best answer to the problem we can with our choice of discretization.

In contrast, the residual error decreases to machine precision levels—this is indicating that our solution is an exact solution to the discrete equation (to roundoff-error). In practice, we can only monitor the residual error, not the true error, and we hope that small residual error implies a small true error.

We can think of the error in the solution as a superposition of high (short) and low (long) frequency (wavelength) modes. Smoothing works really well to eliminate the short wavelength noise quickly (as the exercise shows), but many iterations are

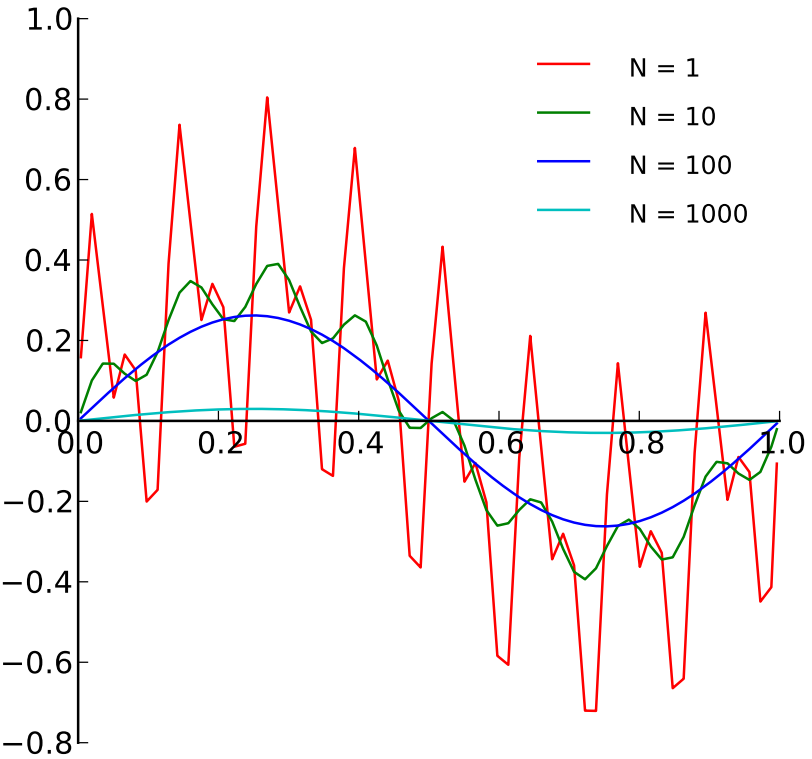

Figure 6.4: Error in the solution to $\phi'' = 0$ given an initial guess with 3 different wavenumbers of noise. The different curves are different numbers of smoothing iterations.

needed to remove the long wavelength noise (see Figure 6.4). Here the wavelength is in terms of the number of zones across the feature, and not a physical measure.

Exercise 6.1: *Implement 1-d smoothing for the Laplace equation on cc-grid. Use an initial guess for the solution:*

$$\phi_0(x) = \frac{1}{3}(\sin(2\pi x) + \sin(2\pi\, 8x) + \sin(2\pi\, 16x)) \tag{6.40}$$

on a 128 zone grid with Dirichlet boundary conditions. This initial guess has both high-frequency and low-frequency noise. Observe that the high-frequency stuff goes after only a few smoothing iterations, but many iterations are needed to remove the low-frequency noise. You should see something like Figure 6.4.

This behavior suggests that if we could represent our problem on a coarser grid, the error will now be of shorter wavelength, and smoothing will be more efficient. This is the core idea behind multigrid.

6.4 Multigrid

The text *A Multigrid Tutorial* [14] provides an excellent introduction to the mechanics of multigrid. The basic idea is to smooth a little on the current grid solving $L\phi = f$, compute the residual, r, then *restrict* r to a coarser grid and smooth on that grid solving $Le = r$, restrict again, ... Once you reach a sufficiently coarse grid, the problem solved exactly. Then the data is moved up to the finer grids, a process called *prolongation*. The error on the coarse grid, e, is prolonged to the finer grid. This error is then used to correct the solution on the finer grid, some smoothing is done, and then the data is prolonged up again.

Note: on the coarse grids, you are not solving the original system, but rather an error equation. If the boundary conditions in the original system are inhomogeneous, the boundary conditions for the error equations are now homogeneous. This must be understood by any ghost cell filling routines.

There are many different forms of the multigrid process. The simplest is called the *V-cycle*. Here you start of the fine grid, restrict down to the coarsest, solve, and then prolong back up to the finest. The flow looks like a 'V'. You continue with additional V-cycles until the residual error is smaller than your tolerance.

6.4.1 Prolongation and restriction on cell-centered grids

Multigrid relies on transferring the problem up and down a hierarchy of grids. Consider the following grid. The finer grid is superposed over the center coarse cell, and the fine grid cells are marked in red.

Restriction from the fine grid to the coarse grid is straightforward. Since the fine cells are perfectly enclosed by a single coarse cell, we simply average:

$$\phi^c_{i,j} = \frac{1}{4}(\phi^f_{--} + \phi^f_{+-} + \phi^f_{-+} + \phi^f_{++}) \tag{6.41}$$

Prolongation requires us to reconstruct the coarse data and use this reconstruction to determine what the fine cell values are. For instance, a linear reconstruction of the coarse data in x and y is:

$$\phi(x,y) = \frac{m_x}{\Delta x}(x - x^c_i) + \frac{m_y}{\Delta y}(y - y^c_j) + \phi^c_{i,j} \tag{6.42}$$

with slopes:

$$m_x = \frac{1}{2}(\phi^c_{i+1,j} - \phi^c_{i-1,j}) \tag{6.43}$$

$$m_y = \frac{1}{2}(\phi^c_{i,j+1} - \phi^c_{i,j-1}) \tag{6.44}$$

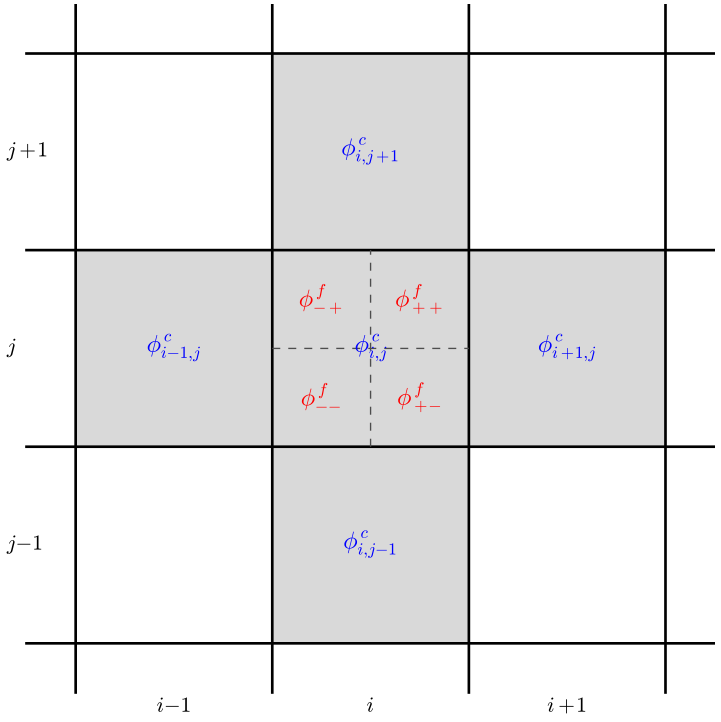

Figure 6.5: Four fine cells and the underlying coarse grid. For prolongation, the fine cells in red are initialized from a coarse parent. The gray coarse cells are used in the reconstruction of the coarse data. For restriction, the fine cells are averaged to the underlying coarse cell.

When averaged over the coarse cell, $\phi(x,y)$ recovers the average, $\phi^c_{i,j}$ in that cell (this means that our interpolant is conservative). We can evaluate the value in the fine cells by evaluating $\phi(x,y)$ at the center of the fine cells,

$$x^f_\pm = x^c_i \pm \frac{\Delta x^c}{4} \tag{6.45}$$

$$y^f_\pm = y^c_j \pm \frac{\Delta y^c}{4} \tag{6.46}$$

$$\tag{6.47}$$

This gives

$$\phi^f_{\pm\pm} = \phi^c_{i,j} \pm \frac{1}{4} m_x \pm \frac{1}{4} m_y \tag{6.48}$$

(Note: you would get the same expression if you averaged $\phi(x,y)$ over the fine cell.)

There are other options for prolongation and restriction, both of higher and lower order accuracy. However, the methods above seem to work well.

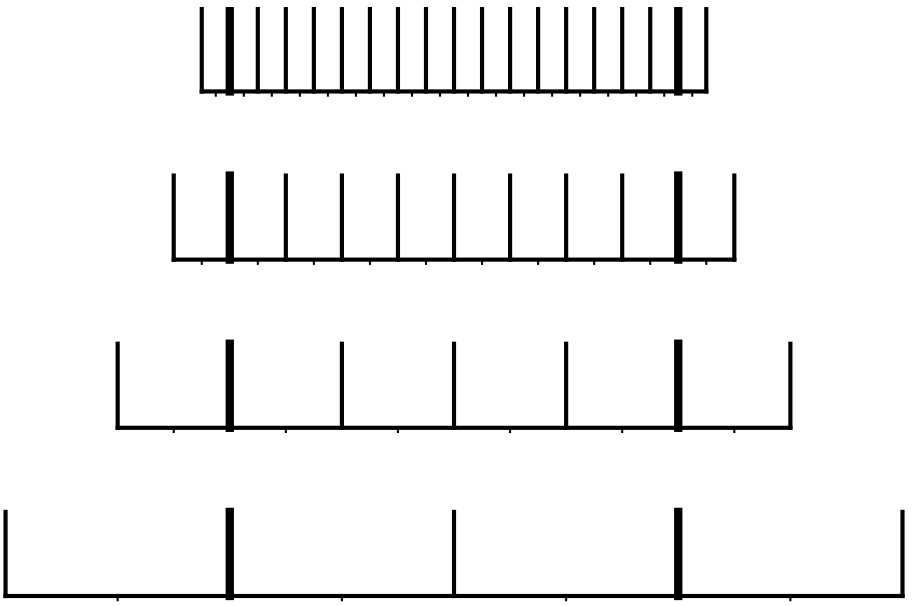

Figure 6.6: Illustration of the hierarchy of grids leading to the coarsest 2-zone grid (in one-dimension). Each grid has a single ghost cell to accommodate boundary conditions.

6.4.2 Bottom solver

Once the grid is sufficiently coarse, the linear system is small enough to be solved directly. This is the bottom solver operation. In the most ideal case, where the finest grid is some power of 2, $N_x = N_y = 2^n$, then the multigrid procedure can continue down until a 2×2 grid is created (Figure 6.6 illustrates this idea for a one-dimensional grid). This is the coarsest grid upon which one can still impose boundary conditions. With this small grid, just doing additional smoothing is sufficient enough to 'solve' the problem. No fancy bottom solver is needed.

For a general rectangular grid or one that is not a power of 2, the coarsest grid will likely be larger. For the general case, a linear system solver like conjugate gradient (or a variant) is used on the coarsest grid.

6.4.3 Boundary conditions throughout the hierarchy

The general inhomogeneous boundary conditions from Eqs. 6.27 and 6.31 apply to the finest level. But because we are solving the residual equation of the coarsest levels in the multigrid hierarchy, the boundary conditions on $Le = r$ are all

homogeneous (but of the same type, Dirichlet, Neumann, or periodic, as the fine level).

Implementing these boundary conditions in your multigrid solver means that you will have separate actions for the fine level (where inhomogeneous boundaries may apply) and the coarser levels (where you will always be homogeneous).

6.4.4 Stopping criteria

Repeated V-cycles are done until:

$$\|r\| < \epsilon \|f\| \tag{6.49}$$

on the finest grid, for some user-input tolerance, ϵ. Here, $\|f\|$ is called the *source norm*. If $\|f\| = 0$, then we stop when

$$\|r\| < \epsilon \tag{6.50}$$

Picking the tolerance ϵ is sometimes problem-dependent, and generally speaking, a problem with a large number of zones will require a looser tolerance.

The general rule-of-thumb is that each V-cycle should reduce your residual by about 1 order of magnitude. It is important that your bottom solver solves the coarse problem to a tolerance of 10^{-3} or 10^{-4} in order for the solver to converge. Figure 6.7 shows the true and residual errors for $\phi_{xx} = \sin(x)$ as a function of V-cycle number, illustrating the expected performance.

The overall convergence of the multigrid algorithm is limited by the discretization of the Laplacian operator used and the implementation of the boundary conditions. Figure 6.8 shows the error in the solution as the number of zones is increased—demonstrating second-order convergence for our implementation.

6.5 Going Further

6.5.1 Red-black Ordering

When using domain decomposition to spread the problem across parallel processors, the smoothing is often done as *red-black Gauss-Seidel*. In this ordering, you imagine the grid to be a checkerboard (see Figure 6.9). In the first Gauss-Seidel pass you update the red squares and in the second, the black squares. The advantage is that when updating the red, you can be sure that none of the zones you depend on (the neighboring black zones) will change. This makes the decomposition parallel. Note: this works for the standard 5-point Laplacian. If you are doing some other operator with a different stencil, then this decomposition may no longer hold.

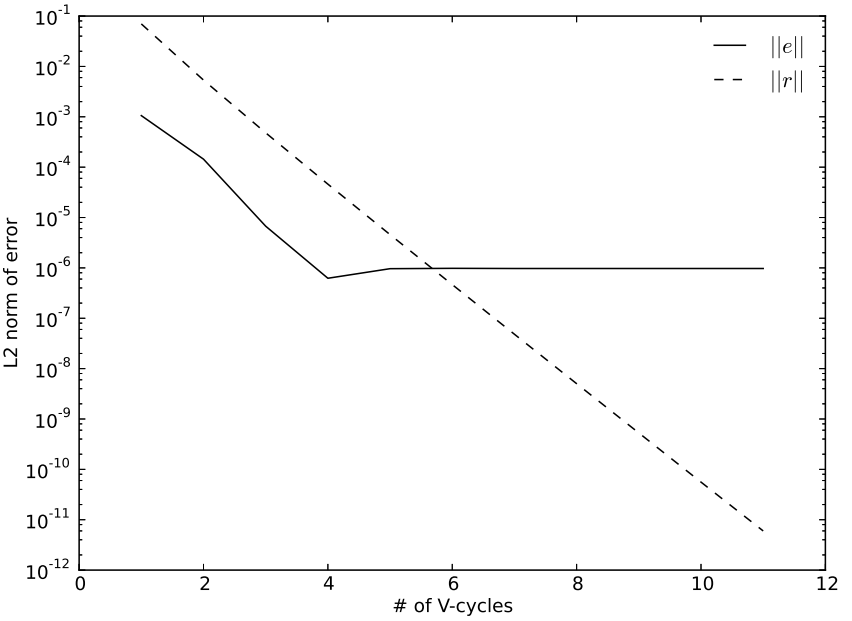

Figure 6.7: Error in the multigrid solution to our model problem ($\phi_{xx} = \sin(x)$) as a function of V-cycle. We see that the true error, $\|e\|$ stalls at truncation error while the residual error, $\|r\|$ reaches roundoff error, the same behavior as seen with smoothing alone (as expected).
hydro_examples: `mg_test.py`

6.5.2 Solvability

For $\nabla^2\phi = f$ with periodic or Neumann boundaries all around, the sum of f must equal 0 otherwise the solution will not converge. Instead, we will simply find the solution increase each V-cycle. This is seen as follows:

$$\int_\Omega f d\Omega = \int_\Omega \nabla^2\phi d\Omega = \int_{\partial\Omega} \nabla\phi \cdot n dS = 0 \tag{6.51}$$

For all homogeneous Neumann boundaries, we have $\nabla\phi \cdot dS = 0$ by construction, so that integral is zero, requiring that the source integrate to zero. If the Neumann boundaries are inhomogeneous, there is still a solvability condition on f based on the sum on the boundary values.

For all periodic boundaries, we have $\nabla\phi|_{\text{left}} = -\nabla\phi|_{\text{right}}$ on the left and right boundaries by definition of the periodicity (and similarly for the top and bottom). Again this implies that f must integrate to zero.

Sometimes, with periodic boundary conditions all around, you need to enforce that f integrate to zero numerically to test convergence. This is discussed in § 10.2.1.

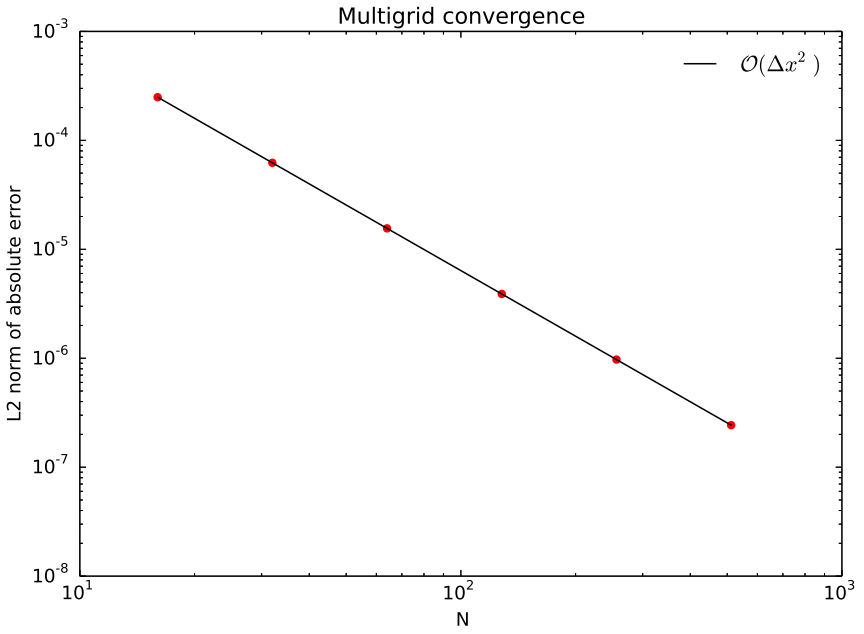

Figure 6.8: Convergence of the multigrid algorithm.
hydro_examples: `mg_converge.py`

6.5.3 Boundary charges

For inhomogeneous boundary conditions, *boundary charges* can be used to convert the BCs to homogeneous BCs. This has the advantage of allowing the ghost cell filling routines only deal with the homogeneous case.

Consider the one-dimensional Poisson equation, near the left boundary our discretized equation appears as:

$$\frac{\phi_{\mathrm{lo}-1} - 2\phi_{\mathrm{lo}} + \phi_{\mathrm{lo}+1}}{\Delta x^2} = f_{\mathrm{lo}} \tag{6.52}$$

Inhomogeneous BCs at the left boundary would give the condition:

$$\phi_{\mathrm{lo}-1} = 2\phi_l - \phi_{\mathrm{lo}} \tag{6.53}$$

Substituting this into the discrete equation, we have:

$$\frac{2\phi_l - \phi_{\mathrm{lo}} - 2\phi_{\mathrm{lo}} + \phi_{\mathrm{lo}+1}}{\Delta x^2} = f_{\mathrm{lo}} \tag{6.54}$$

Bringing the boundary condition value over to the RHS, we see

$$\frac{-3\phi_{\mathrm{lo}} + \phi_{\mathrm{lo}+1}}{\Delta x^2} = f_{\mathrm{lo}} - \frac{2\phi_l}{\Delta x^2} \tag{6.55}$$

Now the left side looks precisely like the differenced Poisson equation with homogeneous Dirichlet BCs. The RHS has an additional 'charge' that captures the

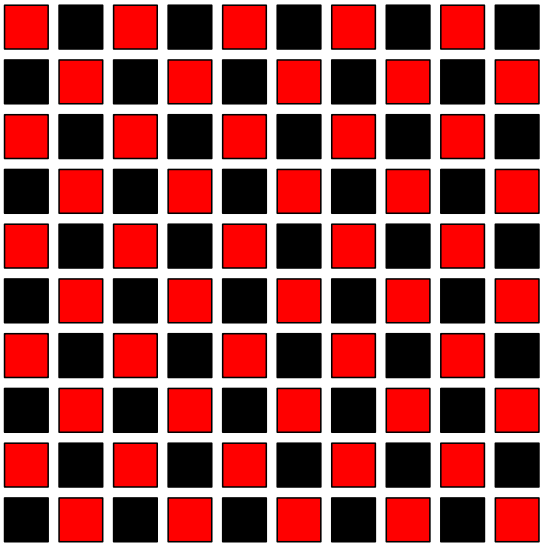

Figure 6.9: The red-black ordering of zones.

boundary value. By modifying the source term, f, in the multigrid solver to include this charge, we can use the homogeneous ghost cell filling routines throughout the multigrid algorithm. This technique is discussed a bit in [21].

Note that the form of the boundary charge will depend on the form of the elliptic equation—the expressions derived above apply only for $\nabla^2\phi = f$.

6.5.4 Norms

There are several different norms that are typically used in defining errors on the grid. The L_∞ norm (or 'inf'-norm) is just the maximum error on the grid:

$$\|e\|_\infty = \max\{|e_{i,j}|\} \tag{6.56}$$

This will pick up on local errors.

The L_1 norm and L_2 norms are more global.

$$\|e\|_1 = \frac{1}{N}\sum_{i,j}|e_{i,j}| \tag{6.57}$$

$$\|e\|_2 = \left(\frac{1}{N}\sum_{i,j}|e_{i,j}|^2\right)^{1/2} \tag{6.58}$$

Generally, the measure in L_2 falls between L_∞ and L_1. Regardless of the norm used, if the problem converges, it should converge in all norms.

For reference, the BoxLib library uses L_∞ in its multigrid solvers.

6.5.5 More General Elliptic Equations

The most general *second-order* elliptic equation takes the form:

$$\alpha\phi + \nabla \cdot (\beta \nabla \phi) + \gamma \cdot \nabla \phi + \nabla \cdot (\zeta \phi) = f \tag{6.59}$$

Here, γ and ζ are vectors. Solving a general elliptic equation of this form can be accomplished with multigrid using the same basic ideas here. The main change is that the smoothing algorithm and the construction of the residual will need to discretize the more general operator, and these coefficients will need to be restricted to the coarser grids (some on edges). This is explored in § 10.2 for a variable-coefficient Poisson equation:

$$\nabla \cdot (\beta \nabla \phi) = f \tag{6.60}$$

and in § 11.3.1 for an equation with $\zeta = 0$.

Chapter 7

Diffusion

These summarize methods for solving the diffusion equation.

7.1 Parabolic equations

The diffusion equation is

$$\frac{\partial \phi}{\partial t} = \frac{\partial}{\partial x}\left(k \frac{\partial \phi}{\partial x}\right) \tag{7.1}$$

This can describe thermal diffusion (for example, as part of the energy equation in compressible flow), species/mass diffusion for multi-species flows, or the viscous terms in incompressible flows. In this form, the diffusion coefficient (or conductivity), k, can be a function of x, or even ϕ. We will consider a constant diffusion coefficient:

$$\frac{\partial \phi}{\partial t} = k \frac{\partial^2 \phi}{\partial x^2} \tag{7.2}$$

The diffusion equation is the prototypical parabolic PDE. The basic behavior of the diffusion equation is to take strongly peaked concentrations of ϕ and smooth them out with time.

7.2 Explicit differencing

The simplest way to difference this equation is *explicit* in time (i.e. the righthand side depends only on the old state):

$$\frac{\phi_i^{n+1} - \phi_i^n}{\Delta t} = k \frac{\phi_{i+1}^n - 2\phi_i^n + \phi_{i-1}^n}{\Delta x^2} \tag{7.3}$$

This is second-order accurate in space, but only first order accurate in time (since the righthand side is not centered in time).

As with the advection equation, when differenced explicitly, there is a constraint on the timestep required for stability. Looking at the growth of a single Fourier mode, $\phi = A^n e^{ij\theta}$ with $j = \sqrt{-1}$, we find:

$$\frac{A^{n+1}}{A^n} = 1 + 2\frac{k\Delta t}{\Delta x^2}(\cos\theta - 1) \tag{7.4}$$

Stability requires that $|A^{n+1}/A^n| \leq 1$, which can only be true if $2k\Delta t/\Delta x^2 \leq 1$. Therefore, our timestep constraint in this case is

$$\Delta t < \frac{1}{2}\frac{\Delta x^2}{k} \tag{7.5}$$

Note the Δx^2 dependence—this constraint can become really restrictive.

To complete the solution, we need boundary conditions at the left (x_l) and right (x_r) boundaries. These are typically either Dirichlet:

$$\phi|_{x=x_l} = \phi_l \tag{7.6}$$
$$\phi|_{x=x_r} = \phi_r \tag{7.7}$$

or Neumann:

$$\phi_x|_{x=x_l} = \phi_x|_l \tag{7.8}$$
$$\phi_x|_{x=x_r} = \phi_x|_r \tag{7.9}$$

7.3 Implicit with direct solve

A backward-Euler implicit discretization would be:

$$\frac{\phi_i^{n+1} - \phi_i^n}{\Delta t} = k\frac{\phi_{i+1}^{n+1} - 2\phi_i^{n+1} + \phi_{i-1}^{n+1}}{\Delta x^2} \tag{7.10}$$

This is still first-order in time, but is not restricted by the timestep constraint (although the timestep will still determine the accuracy). Defining:

$$\alpha \equiv k\frac{\Delta t}{\Delta x^2} \tag{7.11}$$

we can write this as:

$$-\alpha\phi_{i+1}^{n+1} + (1+2\alpha)\phi_i^{n+1} - \alpha\phi_{i-1}^{n+1} = \phi_i^n \tag{7.12}$$

This is a set of coupled algebraic equations. We can write this in matrix form. Using a cell-centered grid, we will solve for the values [lo, hi].

We can define a diffusive CFL number, C, as the ratio of Δt to the maximum explicit timestep (Eq. 7.5), and we see:

$$C \equiv \frac{\Delta t}{\frac{\Delta x^2}{2k}} = 2\alpha \tag{7.13}$$

The implicit method can use any $C > 0$.

We specify boundary conditions by modifying the stencil (Eq. 7.12) for the updates to lo and hi. For example, Neumann BCs on the left mean:

$$\phi_{\text{lo}-1} = \phi_{\text{lo}} \tag{7.14}$$

and substituting this into Eq 7.12, the update for the leftmost cell is:

$$(1+\alpha)\phi_{\text{lo}}^{n+1} - \alpha\phi_{\text{lo}+1}^{n+1} = \phi_{\text{lo}}^{n} \tag{7.15}$$

If we choose Dirichlet BCs on the right ($\phi|_{x=x_l} = A$), then:

$$\phi_{\text{hi}+1} = 2A - \phi_{\text{hi}} \tag{7.16}$$

Substituting this into Eq 7.12 the update for the rightmost cell is:

$$-\alpha\phi_{\text{hi}-1}^{n+1} + (1+3\alpha)\phi_{\text{hi}}^{n+1} = \phi_{\text{hi}}^{n} + \alpha 2A \tag{7.17}$$

For all other interior cells, the stencil is unchanged. The resulting system can be written in matrix form and appears as a *tridiagonal* matrix.

$$\begin{pmatrix} 1+\alpha & -\alpha & & & & & \\ -\alpha & 1+2\alpha & -\alpha & & & & \\ & -\alpha & 1+2\alpha & -\alpha & & & \\ & & \ddots & \ddots & \ddots & & \\ & & & \ddots & \ddots & \ddots & \\ & & & & -\alpha & 1+2\alpha & -\alpha \\ & & & & & -\alpha & 1+3\alpha \end{pmatrix} \begin{pmatrix} \phi_{\text{lo}}^{n+1} \\ \phi_{\text{lo}+1}^{n+1} \\ \phi_{\text{lo}+2}^{n+1} \\ \vdots \\ \vdots \\ \phi_{\text{hi}-1}^{n+1} \\ \phi_{\text{hi}}^{n+1} \end{pmatrix} = \begin{pmatrix} \phi_{\text{lo}}^{n} \\ \phi_{\text{lo}+1}^{n} \\ \phi_{\text{lo}+2}^{n} \\ \vdots \\ \vdots \\ \phi_{\text{hi}-1}^{n} \\ \phi_{\text{hi}}^{n} + \alpha 2A \end{pmatrix} \tag{7.18}$$

This can be solved by standard matrix operations, using a tridiagonal solvers (for example).

A second-order in time discretization requires us to center the righthand side in time. We do this as:

$$\frac{\phi_i^{n+1} - \phi_i^n}{\Delta t} = \frac{k}{2}\left(\frac{\phi_{i+1}^n - 2\phi_i^n + \phi_{i-1}^n}{\Delta x^2} + \frac{\phi_{i+1}^{n+1} - 2\phi_i^{n+1} + \phi_{i-1}^{n+1}}{\Delta x^2}\right) \tag{7.19}$$

This time-discretization is called *Crank-Nicolson*. Again, using $\alpha \equiv k\Delta t/\Delta x^2$, and grouping all the $n+1$ terms on the left we have:

$$\phi_i^{n+1} - \frac{\alpha}{2}\left(\phi_{i+1}^{n+1} - 2\phi_i^{n+1} + \phi_{i-1}^{n+1}\right) = \phi_i^n + \frac{\alpha}{2}\left(\phi_{i+1}^n - 2\phi_i^n + \phi_{i-1}^n\right) \tag{7.20}$$

and grouping together the the $n+1$ terms by zone, we have:

$$-\frac{\alpha}{2}\phi_{i+1}^{n+1} + (1+\alpha)\phi_i^{n+1} - \frac{\alpha}{2}\phi_{i-1}^{n+1} = \phi_i^n + \frac{\alpha}{2}\left(\phi_{i+1}^n - 2\phi_i^n + \phi_{i-1}^n\right) \tag{7.21}$$

Considering Neumann boundary conditions on the left, we again have $\phi_{\mathrm{lo}-1}^{n+1} = \phi_{\mathrm{lo}}^{n+1}$, and our stencil at the boundary becomes

$$-\frac{\alpha}{2}\phi_{\mathrm{lo}+1}^{n+1} + \left(1+\frac{\alpha}{2}\right)\phi_{\mathrm{lo}}^{n+1} = \phi_{\mathrm{lo}}^n + \frac{\alpha}{2}\left(\phi_{\mathrm{lo}+1}^n - 2\phi_{\mathrm{lo}}^n + \phi_{\mathrm{lo}-1}^n\right) \tag{7.22}$$

The matrix form of this system is:

$$\begin{pmatrix}
1+\frac{\alpha}{2} & -\frac{\alpha}{2} & & & & & \\
-\frac{\alpha}{2} & 1+\alpha & -\frac{\alpha}{2} & & & & \\
 & -\frac{\alpha}{2} & 1+\alpha & -\frac{\alpha}{2} & & & \\
 & & \ddots & \ddots & \ddots & & \\
 & & & \ddots & \ddots & \ddots & \\
 & & & & -\frac{\alpha}{2} & 1+\alpha & -\frac{\alpha}{2} \\
 & & & & & -\frac{\alpha}{2} & 1+\frac{\alpha}{2}
\end{pmatrix}
\begin{pmatrix}
\phi_{\mathrm{lo}}^{n+1} \\ \phi_{\mathrm{lo}+1}^{n+1} \\ \phi_{\mathrm{lo}+2}^{n+1} \\ \vdots \\ \vdots \\ \phi_{\mathrm{hi}-1}^{n+1} \\ \phi_{\mathrm{hi}}^{n+1}
\end{pmatrix}
=
\begin{pmatrix}
\phi_{\mathrm{lo}}^n + \frac{k\Delta t}{2}[\nabla^2\phi]_{\mathrm{lo}}^n \\
\phi_{\mathrm{lo}+1}^n + \frac{k\Delta t}{2}[\nabla^2\phi]_{\mathrm{lo}+1}^n \\
\phi_{\mathrm{lo}+2}^n + \frac{k\Delta t}{2}[\nabla^2\phi]_{\mathrm{lo}+2}^n \\
\vdots \\
\vdots \\
\phi_{\mathrm{hi}-1}^n + \frac{k\Delta t}{2}[\nabla^2\phi]_{\mathrm{hi}-1}^n \\
\phi_{\mathrm{hi}}^n + \frac{k\Delta t}{2}[\nabla^2\phi]_{\mathrm{hi}}^n
\end{pmatrix} \tag{7.23}$$

Figure 7.1 shows the result of using $\alpha = 0.8$ and $\alpha = 8.0$. We see that they are both stable, but that the smaller timestep is closer to the analytic solution (especially at early times).

Exercise 7.1: *Write a one-dimensional implicit diffusion solver for the domain $[0,1]$ with Neumann boundary conditions at each end and $k = 1$. Your solver should use a tridiagonal solver and initialize a matrix like that above. Use a timestep close to the explicit step, a grid with N = 128 zones.*

If we begin with a Gaussian, the resulting solution is also a Gaussian. Initialize using the following with $t = 0$:

$$\phi(x,t) = (\phi_2 - \phi_1)\sqrt{\frac{t_0}{t+t_0}}e^{-\frac{1}{4}(x-x_c)^2/k(t+t_0)} + \phi_1 \tag{7.24}$$

with $t_0 = 0.001$, $\phi_1 = 1$, and $\phi_2 = 2$, and x_c is the coordinate of the center of the domain. Run until $t = 0.01$ and compare to the analytic solution above.

(Note: the solution for two-dimensions differs slightly)

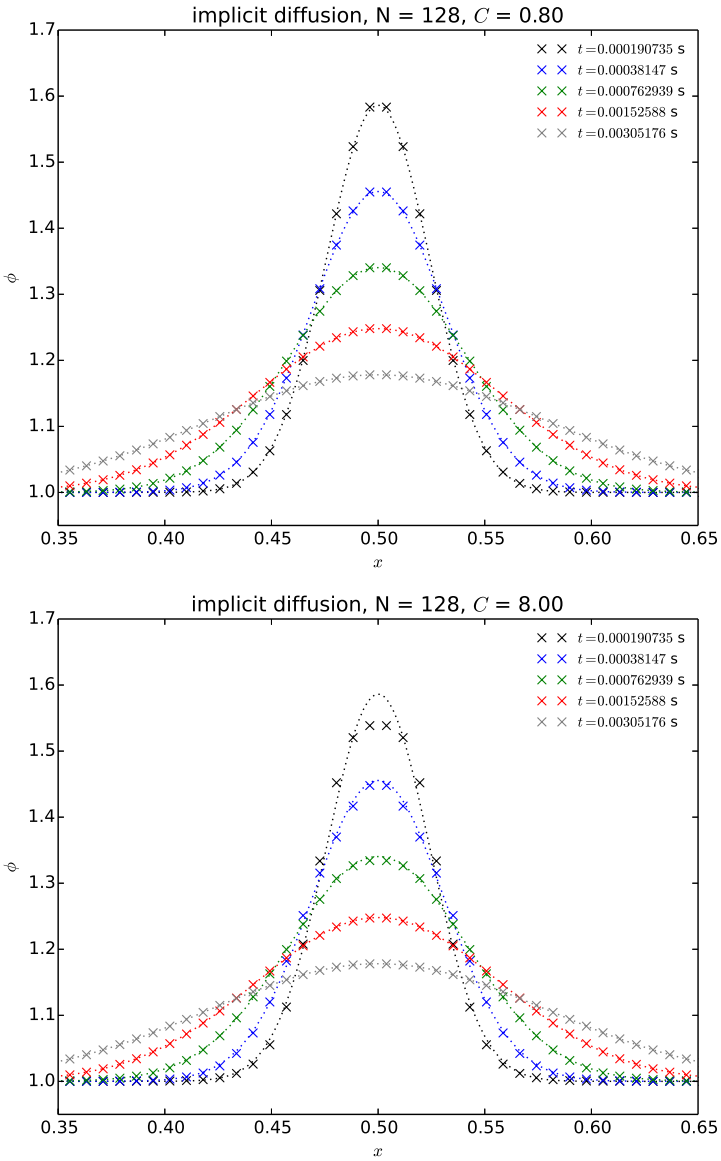

Figure 7.1: Implicit diffusion of a Gaussian (with Crank-Nicolson discretization) with $C = 0.8$ and $C = 8.0$. The exact solution at each time is shown as the dotted line.

hydro_examples: `diffusion-implicit.py`

7.4 Implicit multi-dimensional diffusion via multigrid

Instead of doing a direct solve of the matrix form of the system, we can use multigrid techniques. Consider the Crank-Nicolson system we just looked at:

$$\frac{\phi_i^{n+1} - \phi_i^n}{\Delta t} = \frac{1}{2}\left(k\nabla^2\phi_i^n + k\nabla^2\phi_i^{n+1}\right) \tag{7.25}$$

Grouping all the $n+1$ terms on the left, we find:

$$\phi_i^{n+1} - \frac{\Delta t}{2} k \nabla^2 \phi_i^{n+1} = \phi_i^n + \frac{\Delta t}{2} k \nabla^2 \phi_i^n \tag{7.26}$$

This is in the form of a constant-coefficient Helmholtz equation,

$$(\alpha - \beta \nabla^2)\phi^{n+1} = f \tag{7.27}$$

with

$$\alpha = 1 \tag{7.28}$$

$$\beta = \frac{\Delta t}{2} k \tag{7.29}$$

$$f = \phi^n + \frac{\Delta t}{2} k \nabla^2 \phi^n \tag{7.30}$$

This can be solved using multigrid techniques with a Helmholtz operator. The same boundary conditions described above apply here.

Note: when using multigrid, you do not need to actually construct the matrix. This is usually the most efficient way to implement diffusion in a multi-dimensional simulation code, especially when distributing the grid across parallel processors.

7.5 Going further

- *Non-constant conductivity*: for the case where $k = k(x)$, we discretize as:

$$\frac{\phi_i^{n+1} - \phi_i^n}{\Delta t} = \frac{\{k \nabla \phi\}_{i+1/2} - \{k \nabla \phi\}_{i-1/2}}{\Delta x} \tag{7.31}$$

  Here we need the values of k at the interfaces, $k_{i-1/2}$ and $k_{i+1/2}$. We can get these from the cell-centered values in a variety of ways including straight-averaging:

$$k_{i+1/2} = \frac{1}{2}(k_i + k_{i+1}) \tag{7.32}$$

  or averaging the inverses:

$$\frac{1}{k_{i+1/2}} = \frac{1}{2}\left(\frac{1}{k_i} + \frac{1}{k_{i+1}}\right) \tag{7.33}$$

- *State-dependent transport coefficients*: many times the transport coefficients themselves depend on the quantity being diffused:

$$\frac{\phi_i^{n+1} - \phi_i^n}{\Delta t} = \frac{1}{2}\left\{\nabla \cdot [k(\phi^n)\nabla \phi^n]_i + \nabla \cdot [k(\phi^{n+1})\nabla \phi^{n+1}]_i\right\} \tag{7.34}$$

(for example, with thermal diffusion, the conductivity can be temperature dependent). In this case, we can achieve second-order accuracy by doing a predictor-corrector. First we diffuse with the transport coefficients evaluated at the old time, giving a provisional state, $\phi^\star$:

$$\frac{\phi_i^\star - \phi_i^n}{\Delta t} = \frac{1}{2}\left\{\nabla\cdot[k(\phi^n)\nabla\phi^n] + \nabla\cdot[k(\phi^n)\nabla\phi^\star]\right\} \tag{7.35}$$

Then we redo the diffusion, evaluating k with $\phi^\star$ to center the righthand side in time, giving the new state, ϕ^{n+1}:

$$\frac{\phi_i^{n+1} - \phi_i^n}{\Delta t} = \frac{1}{2}\left\{\nabla\cdot[k(\phi^n)\nabla\phi^n] + \nabla\cdot[k(\phi^\star)\nabla\phi^{n+1}]\right\} \tag{7.36}$$

This is the approach used, for example, in [11].

- *Temperature diffusion in energy equation*: Often we find diffusion represented as one of many physical processes in a single equation. For example, consider the internal energy equation with both reactions and diffusion:

$$\rho\frac{\partial e}{\partial t} + \rho U\cdot\nabla e + p\nabla\cdot U = \nabla\cdot k\nabla T + \rho S \tag{7.37}$$

This can be solved via an explicit-implicit discretization. First the advection terms are computed as:

$$A = \rho U\cdot\nabla e + p\nabla\cdot U \tag{7.38}$$

Then the advective-diffusive part is solved implicitly. Expressing $e = e(\rho, T)$, and rewriting

$$\nabla T = (\nabla e - e_\rho\nabla\rho)/e_T \tag{7.39}$$

and then

$$\rho\frac{\partial e}{\partial t} = \nabla\cdot(k/e_T)\nabla e - \nabla\cdot(ke_\rho/e_T)\nabla\rho - A + \rho S \tag{7.40}$$

This is now a diffusion equation for e, which can be solved by the techniques described above. This is discussed, for example, in [11, 33].

Chapter 8

Multiphysics Applications

Generally, we are interested in multiphysics simulations—these combine a little bit of everything we discussed so far

8.1 Integrating Multiphysics

Consider an equation whose evolution depends on several different physical processes, represented by the operators A, D, R (these may represent, e.g., advection, diffusion, reactions, etc.).

$$\phi_t = -A(\phi) + D(\phi) + R(\phi) \tag{8.1}$$

One way to solve this system is to discretize each of the operators in space. For instance the discrete advection operator, $[A(\phi)]_i$ might use the ideas on piecewise linear reconstruction techniques discussed in chapter 3, $[D(\phi)]_i$ can use the discrete form of the Laplacian from chapter 7, and $[R(\phi)]_i$ may be an algebraic relation. This leaves us with an ordinary differential equation for the time-evolution of ϕ,

$$\frac{d\phi_i}{dt} = -[A(\phi)]_i + [D(\phi)]_i + [R(\phi)]_i \tag{8.2}$$

which can be solve using standard ODE techniques. This technique is called the *method of lines*, and can be a powerful technique to solve PDEs or systems of PDEs with multiple physics operators.

A difficulty arises if these processes each have different timescales associated with them. For instance, reactions may be vigorous and require a small timestep to accurately capture the changes, but the advection is slow. Therefore, we don't want to use the same timestep for all the processes, since that will needlessly make things computationally expensive. *Operator splitting* solves for the effects of each

operator separately, using whichever timestep (and time-discretization, e.g., explicit or implicit) is most suited to the operation. The result of one operation is used as the input to the next. The downside of this approach is that the operations may not be well coupled.

A final technique, *spectral deferred corrections* combines the best features of the method of lines approach and operator splitting.

8.2 Ex: diffusion-reaction

Consider a diffusion-reaction equation:

$$\phi_t = \kappa \phi_{xx} + \frac{1}{\tau} R(\phi) \tag{8.3}$$

This can be thought of as a simple model for a combustion flame, and can propagate a front. It is often the case that the reactions are stiff, and require a smaller timestep then the diffusion part. In fact, we may want to use an implicit integration method designed for stiff ODEs for the reaction part, but use a standard explicit method for the diffusion part. This requires operator splitting.

We can use Strang splitting [49] to make the integration second-order accurate overall:

$$\phi^{n+1} = R_{\Delta t/2} D_{\Delta t} R_{\Delta t/2} \phi^n \tag{8.4}$$

where $R_{\Delta t/2}$ represents reacting for a step of $\Delta t/2$ and $D_{\Delta t}$ represents diffusing for a step of Δt. In each case, these operators act as if the other were not present, but they see the effect of the previous operation on the input ϕ.

No explicit source terms describing one process appear in the other process's update. The procedure for updating appears as:

1. *Evolve reaction ODE system for* $\Delta t/2$

$$\frac{d\phi^\star}{dt} = \frac{1}{\tau} R(\phi^\star), \qquad \phi^\star(0) = \phi^n \tag{8.5}$$

2. *Solve the diffusion equation for* Δt *with an implicit Crank-Nicolson discretization*

$$\frac{\phi^{\star\star} - \phi^\star}{\Delta t} = \frac{1}{2}(D(\phi^\star) + D(\phi^{\star\star})) \tag{8.6}$$

3. *Evolve reaction ODE system for* $\Delta t/2$

$$\frac{d\phi^{n+1}}{dt} = \frac{1}{\tau} R(\phi^{n+1}), \qquad \phi^{n+1}(0) = \phi^{\star\star} \tag{8.7}$$

Figure 8.1 shows the solution to our diffusion-reaction equation with 256 zones, $\kappa = 0.1$, $\tau = 1.0$ at several times.

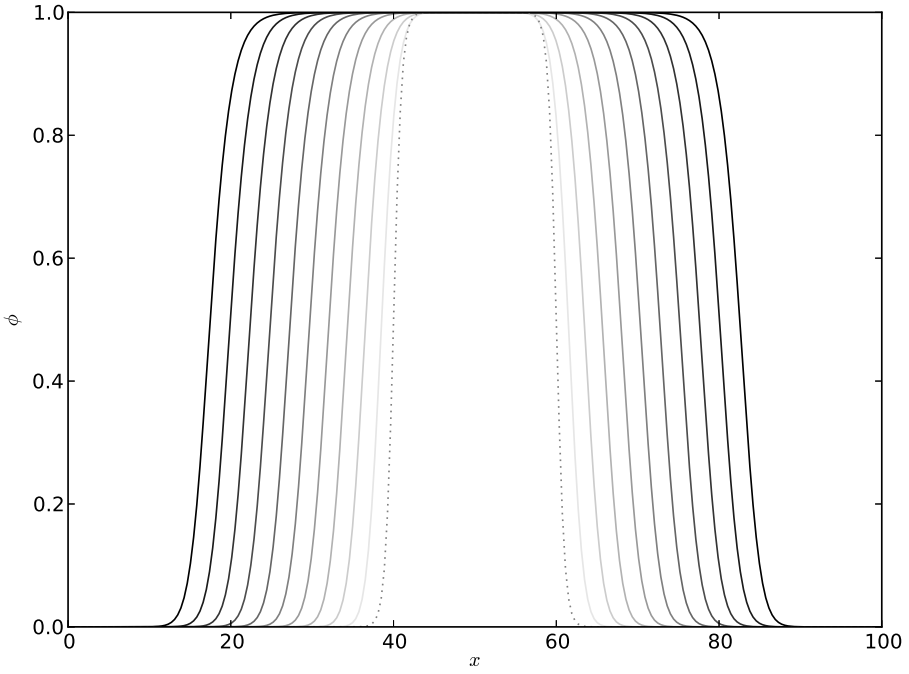

Figure 8.1: Solution to the diffusion-reaction equation with 256 zones, and $\kappa = 0.1$, $\tau = 1.0$. The lines shown are spaced 8.0 time-units apart. We see the initial smoothed tophat profile giving rise to a traveling front.
hydro_examples: `diffusion-reaction.py`

Exercise 8.1: *Consider a simple reaction source*

$$R(\phi) = \frac{1}{4}\phi(1-\phi) \tag{8.8}$$

This is called a KPP reaction source. Here ϕ can be thought of as a progress variable that varies between pure ash ($\phi = 0$) and pure fuel ($\phi = 1$).

Solve the system with this source. Note that you should begin with some smooth initial conditions—if they are too sharp than the C-N discretization will cause jagged features to appear. The solution in this case is a wave with speed $S = \sqrt{\kappa/\tau}$ and thickness $\delta = \sqrt{\kappa\tau}$ (see [53] for some details of this system).

8.3 Ex: advection-diffusion

The viscous Burgers' equation appears as:

$$u_t + uu_x = \epsilon u_{xx} \tag{8.9}$$

This admits shocks and rarefactions just like the inviscid form, but now the viscosity can act to smooth out the shock—instead of being infinitely thin, it will have a physical width.

As we saw earlier, there are efficient, accurate methods for handling the explicit parts explicitly, but for diffusion, we often want to solve it implicitly. We can split the solution up, but couple the two processes together to make a method that is overall second-order accurate in time. We write our equation as:

$$u_t + A(u) = D(u) \tag{8.10}$$

with $A(u) = [\frac{1}{2}u^2]_x$ and $D(u) = eu_{xx}$. Then our update appears in two steps.

1. *Find the advective update over the timestep*: We use an approximation of the diffusion term at time-level n, $D(u^n)$ as a source in the construction of the interface states for the advective part. Once the interface states, $u^{n+1/2}_{i+1/2}$ are known, we construct the advective update term as:

$$A_i^{n+1/2} = \frac{\left[\frac{1}{2}\left(u^{n+1/2}_{i+1/2}\right)^2\right] - \left[\frac{1}{2}\left(u^{n+1/2}_{i-1/2}\right)^2\right]}{\Delta x} \tag{8.11}$$

2. *Solve the diffusion equation with the advective source*: We use a Crank-Nicolson discretization of the diffusion part of our equation, with the advective update term appearing as a source.

$$\frac{u^{n+1} - u^n}{\Delta t} = \frac{1}{2}D(u^n) + \frac{1}{2}D(u^{n+1}) - A^{n+1/2} \tag{8.12}$$

   This is a linear system that can be solved as a tridiagonal matrix or with multigrid. The result of this step is that u^{n+1} is updated with both the advection and diffusion terms.

Because the diffusion is done implicitly, the timestep constraint (for stability) for this solve is due to the advective portion only.

For step 1, the addition of the explicit diffusion source requires a small change to the method we used to predict the interface states.

$$\begin{aligned}
u^{n+1}_{i+1/2,L} &= u^n_i + \frac{\Delta x}{2}\frac{\partial u}{\partial x} + \frac{\Delta t}{2}\frac{\partial u}{\partial t} + \ldots && (8.13)\\
&= u^n_i + \frac{\Delta x}{2}\frac{\partial u}{\partial x} + \frac{\Delta t}{2}\left(-u_i\frac{\partial u}{\partial x} + D(u^n_i)\right) + \ldots && (8.14)\\
&= u^n_i + \frac{\Delta x}{2}\left(1 - \frac{\Delta t}{\Delta x}u_i\right)\frac{\partial u}{\partial x} + \frac{\Delta t}{2}D(u^n) + \ldots && (8.15)
\end{aligned}$$

here the source term (shown in red) incorporates the effects of the diffusion on the prediction of the states for advection. This entered into our states when we

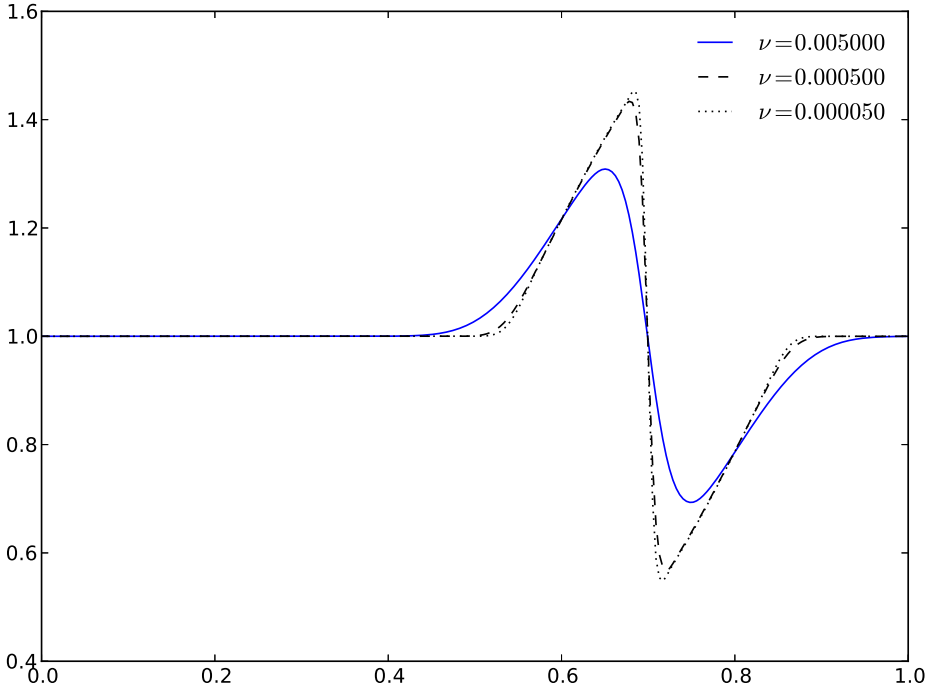

Figure 8.2: Solution to the viscous Burgers' equation with a variety of different viscosities. The initial conditions was a single wavelength of a sine wave for $x \in [1/3, 2/3]$, and $u = 1$ otherwise.
hydro_examples: `burgersvisc.py`

replaced $\partial u/\partial t$ with our PDE equation. The spatial derivative, $\partial u/\partial x$ is replaced by a monotonized difference, and the method then proceeds as with the regular Burgers' equation. The Riemann problem is unchanged from the inviscid case.

Figure 8.2 shows the solution of the viscous Burgers' equation for shock initial conditions with different amounts of viscosity. Notice that the effect of the viscosity is to smooth the shock profile, but the shock position itself agrees between the cases.

Chapter 9

Incompressible Flow and Projection Methods

These summarize methods for solving the incompressible hydrodynamics equations using an cell-centered approximate projection method.

9.1 Incompressible flow

As a fluid parcel advects through a domain, it compresses and expands due to a variety of effects (stratification, local heat release, acoustic/shock waves). The Lagrangian derivative of the density captures the changes in the fluid, and is a measure of its compressibility. From the continuity equation, we see:

$$-\frac{1}{\rho}\frac{D\rho}{Dt} = \nabla \cdot U \tag{9.1}$$

Note that for $\nabla \cdot U > 0$, we have $-(1/\rho)(D\rho/Dt) > 0$, which means that ρ gets smaller—this is expansion of the Lagrangian fluid element.

A fluid in which the density (and therefore volume) of a fluid element is not allowed to change is called *incompressible*. An incompressible fluid obeys the velocity constraint:

$$\nabla \cdot U = 0 \tag{9.2}$$

(since $D\rho/Dt = 0$). The incompressible fluid approximation is a reasonable approximation when the Mach number of the fluid is small ($\ll 1$). To complete the system, we add the momentum equation. If we take the density to be constant everywhere in the domain (not just in our fluid element), then we have:

$$\frac{\partial U}{\partial t} + U \cdot \nabla U + \nabla p = 0 \tag{9.3}$$

Note that p here looks like a pressure, but it is not subject to any equation of state. This system is closed as written. The value of p is determined such that the velocity constraint is satisfied.

9.2 Projection methods

The basic idea behind a projection method is that any vector field can be decomposed into a divergence free part and the gradient of a scalar (this is sometimes called a *Hodge decomposition*). Given a velocity field $U^\star$, we can express it in terms of the divergence free part U^d and a scalar, ϕ as:

$$U^\star = U^d + \nabla\phi \tag{9.4}$$

Taking the divergence of each side, and noting that $\nabla \cdot U^d = 0$, we have

$$\nabla \cdot U^\star = \nabla^2\phi \tag{9.5}$$

This is an elliptic equation. Given suitable boundary conditions, we can solve for ϕ (for instance, using multigrid) and recover the divergence free part of $U^\star$ as:

$$U^d = U^\star - \nabla\phi \tag{9.6}$$

We call this operation of extracting the divergence free part of a velocity field a *projection*. This can be expressed by defining an operator P, such that $PU^\star = U^d$, and $(I - P)U^\star = \nabla\phi$. From the momentum equation, we see that $\partial U/\partial t + \nabla\phi$ is in the form of a divergence free term + the gradient of a scalar. This means that advancing the velocity field subject to the constraint involves solving:

$$U_t = P(U_t + \nabla p) = P(-U \cdot \nabla U) \tag{9.7}$$

See Bell, Colella, and Howell [9] for a nice discussion of this.

The original projection method for incompressible flows goes back to Chorin [17]. Instead of evolving Eq. 9.7 directly, we break the update into pieces. The basic idea is to evolve the velocity advection equation without regard to the constraint, yielding a *provisional velocity* field which is then subjected to a projection to enforce the divergence-free constraint. Bell, Colella, and Glaz (BCG) [8] introduced a projection method that uses standard Godunov methods for the advection terms (much like is done with compressible flow) and then solves an elliptic equation to enforce the constraint. This division of the operations in the algorithm (advect then project) is a type of *fractional step* method.

There are different ways to discretize the operators that make up the projection. We denote the discretized divergence as D and the discretized gradient operation as G. For an exact projection, the discretized Laplacian, L, would be the same as applying G and D in succession (i.e. $L = DG$). Depending on our data centerings, we may prefer to discretize the Laplacian independently to D and G, such that

$L \neq DG$. This is called an *approximate projection*. Note that for an approximate projection, P is not idempotent: $P^2U \neq PU$.

Many variations on this basic idea exist, using alternate forms of the projection, different grid centerings of the ϕ variable, and additional physics.

9.3 Cell-centered approximate projection solver

Here we describe an incompressible algorithm that uses cell-centered data throughout—U and p are both cell-centered. The projection at the end is an approximate projection. The basic algorithm flow is

- Create the time-centered advective velocities through the faces of the zones.
- Project the advective velocities such that they obey the velocity constraint
- Construct the time-centered interface states of all quantities on the faces of the zones using the advective velocity.
- Update the velocity to the new time. This is defines the provisional velocity field—it does not yet satisfy the constraint.
- Enforce the velocity constraint by projecting the velocity.

The description below is pieced together from a variety of sources. BCH describes a cell-centered method, but with an exact projection (with a larger, decoupled stencil). Almgren, Bell, and Szymczak (ABS) [7] describes an approximate projection method, but with a node-centered final projection. We follow this paper closely up until the projection. Martin and Colella [34] (and Martin's PhD thesis) method uses a cell-centered projection, as is described here. They go into additional effort to describe this for a refined grid. All of these method are largely alike, aside from how the discretization of the final projection is handled.

9.3.1 Advective velocity

In predicting the interface states, we first seek to construct the velocities through the interfaces. A key concept throughout the advection step is that once we have the normal velocities on the interfaces, we can use these to upwind left and right states of any quantity to get their interface value. The advective velocities we construct here, u^{adv} and v^{adv}, will later be used to upwind the various states we predict to the interfaces. We only need the velocity through the interfaces, as shown in the figure 9.1. This staggered grid arrangement is sometimes called a MAC grid.

We follow ABS. Our velocity evolution system (writing out the individual compo-

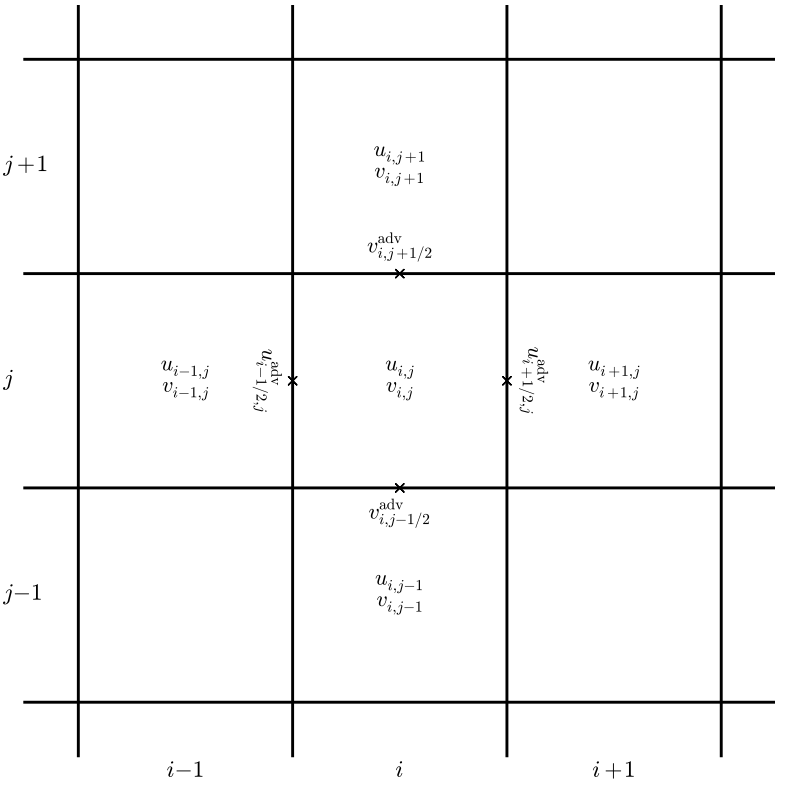

Figure 9.1: The staggered 'MAC' grid for the advective velocities.

nents of U: u and v) is

$$\frac{\partial u}{\partial t} = -u\frac{\partial u}{\partial x} - v\frac{\partial u}{\partial y} - \frac{\partial p}{\partial x} = 0 \tag{9.8}$$

$$\frac{\partial v}{\partial t} = -u\frac{\partial v}{\partial x} - v\frac{\partial v}{\partial y} - \frac{\partial p}{\partial y} = 0 \tag{9.9}$$

Our goal in this step is to predict time-centered interface values of the normal velocity (u on x-edges and v on y-edges). The prediction follows from Taylor expanding the state to the interface (through $\Delta x/2$ or $\Delta y/2$) and to the half-time (through $\Delta t/2$). As with the regular advection, we can have left and right states which we will resolve by solving a Riemann problem. The left interface state of u

at $i+1/2, j$ is found as:

$$u^{n+1/2}_{i+1/2,j,L} = u_{i,j} + \frac{\Delta x}{2}\left.\frac{\partial u}{\partial x}\right|_{i,j} + \left.\frac{\Delta t}{2}\frac{\partial u}{\partial t}\right|_{i,j} \tag{9.10}$$

$$= u_{i,j} + \frac{\Delta x}{2}\left.\frac{\partial u}{\partial x}\right|_{i,j} + \frac{\Delta t}{2}\left.\left(-u\frac{\partial u}{\partial x} - v\frac{\partial u}{\partial y} - \frac{\partial p}{\partial x}\right)\right|_{i,j} \tag{9.11}$$

$$= u_{i,j} + \frac{\Delta x}{2}\left(1 - \frac{\Delta t}{\Delta x}u_{i,j}\right)\left.\frac{\partial u}{\partial x}\right|_{i,j} - \frac{\Delta t}{2}\left(v\frac{\partial u}{\partial y}\right)_{i,j} - \left.\frac{\Delta t}{2}\frac{\partial p}{\partial x}\right|_{i,j} \tag{9.12}$$

$$\tag{9.13}$$

We express $\partial u/\partial x|_{i,j}$ as $\overline{\Delta u}^{(x)}_{i,j}/\Delta x$, where $\overline{\Delta u}^{(x)}_{i,j}$ is the limited slope of u in the x-direction in zone i, j. Our interface state is then:

$$u^{n+1/2}_{i+1/2,j,L} = \underbrace{u_{i,j} + \frac{1}{2}\left(1 - \frac{\Delta t}{\Delta x}u\right)\overline{\Delta u}^{(x)}_{i,j}}_{\equiv \hat{u}^{n+1/2}_{i+1/2,j,L}} - \underbrace{\frac{\Delta t}{2}\left(v\frac{\partial u}{\partial y}\right)_{i,j}}_{\text{transverse term}} - \left.\frac{\Delta t}{2}\frac{\partial p}{\partial x}\right|_{i,j} \tag{9.14}$$

Similarly, for v through the y faces, we find:

$$v^{n+1/2}_{i,j+1/2,L} = \underbrace{v_{i,j} + \frac{1}{2}\left(1 - \frac{\Delta t}{\Delta x}v\right)\overline{\Delta v}^{(y)}_{i,j}}_{\equiv \hat{v}^{n+1/2}_{i,j+1/2,L}} - \underbrace{\frac{\Delta t}{2}\left(u\frac{\partial v}{\partial x}\right)_{i,j}}_{\text{transverse term}} - \left.\frac{\Delta t}{2}\frac{\partial p}{\partial y}\right|_{i,j} \tag{9.15}$$

As indicated above (and following ABS and the similar notation used by Colella [19]), we denote the quantities that will be used to evaluate the transverse states (consisting only of the normal predictor) with a '^'. These will be used to evaluate the transverse terms labeled above.

We predict u and v to both the x and y interfaces, using only the normal part of the predictor. This gives us the left and right 'hat' states on each interface.

u on x-interfaces:

$$\hat{u}^{n+1/2}_{i+1/2,j,L} = u_{i,j} + \frac{1}{2}\left(1 - \frac{\Delta t}{\Delta x}u_{i,j}\right)\overline{\Delta u}^{(x)}_{i,j} \tag{9.16}$$

$$\hat{u}^{n+1/2}_{i+1/2,j,R} = u_{i+1,j} - \frac{1}{2}\left(1 + \frac{\Delta t}{\Delta x}u_{i+1,j}\right)\overline{\Delta u}^{(x)}_{i+1,j} \tag{9.17}$$

v on x-interfaces:

$$\hat{v}^{n+1/2}_{i+1/2,j,L} = v_{i,j} + \frac{1}{2}\left(1 - \frac{\Delta t}{\Delta x}u_{i,j}\right)\overline{\Delta v}^{(x)}_{i,j} \tag{9.18}$$

$$\hat{v}^{n+1/2}_{i,j+1/2,R} = v_{i+1,j} - \frac{1}{2}\left(1 + \frac{\Delta t}{\Delta x}u_{i+1,j}\right)\overline{\Delta v}^{(x)}_{i+1,j} \tag{9.19}$$

u on y-interfaces:

$$\hat{u}^{n+1/2}_{i,j+1/2,L} = u_{i,j} + \frac{1}{2}\left(1 - \frac{\Delta t}{\Delta y} v_{i,j}\right) \overline{\Delta u}^{(y)}_{i,j} \tag{9.20}$$

$$\hat{u}^{n+1/2}_{i,j+1/2,R} = u_{i,j+1} - \frac{1}{2}\left(1 + \frac{\Delta t}{\Delta y} v_{i,j+1}\right) \overline{\Delta u}^{(y)}_{i,j+1} \tag{9.21}$$

v on y-interfaces:

$$\hat{v}^{n+1/2}_{i,j+1/2,L} = v_{i,j} + \frac{1}{2}\left(1 - \frac{\Delta t}{\Delta y} v_{i,j}\right) \overline{\Delta v}^{(y)}_{i,j} \tag{9.22}$$

$$\hat{v}^{n+1/2}_{i,j+1/2,R} = v_{i,j+1} - \frac{1}{2}\left(1 + \frac{\Delta t}{\Delta y} v_{i,j+1}\right) \overline{\Delta v}^{(y)}_{i,j+1} \tag{9.23}$$

Note that the 'right' state is constructed using the data to the right of the interface. Also note that in constructing these transverse velocities, we do not include the p term.

Next we find the advective velocity through each interface. The incompressible velocity equation looks like the inviscid Burger's equation, and the Riemann solver follows that construction. BCG provide the implementation used here (and throughout the incompressible literature). Also see Toro [51]. We denote the resulting velocities with the 'adv' superscript, as these are the normal velocities used to advect the hat states. The Riemann problem solution is:

$$\mathcal{R}(q_L, q_R) = \begin{cases} q_L & \text{if } q_L > 0, \quad q_L + q_R > 0 \\ 0 & \text{if } q_L \leq 0, \quad q_R \geq 0 \\ q_R & \text{otherwise} \end{cases} \tag{9.24}$$

We solve this for each of the normal velocities, giving:

$$\hat{u}^{\text{adv}}_{i+1/2,j} = \mathcal{R}(\hat{u}^{n+1/2}_{i+1/2,j,L}, \hat{u}^{n+1/2}_{i+1/2,j,R}) \tag{9.25}$$

$$\hat{v}^{\text{adv}}_{i,j+1/2} = \mathcal{R}(\hat{v}^{n+1/2}_{i,j+1/2,L}, \hat{v}^{n+1/2}_{i,j+1/2,R}) \tag{9.26}$$

These advective velocities (sometimes called the *transverse velocities*) are used to resolve the left and right states of all the hat quantities by simple upwinding. For a u or v state on the x-interface, we upwind based on $\hat{u}^{\text{adv}}$; and for a u or v state on the y-interface, we upwind based on $\hat{v}^{\text{adv}}$. If we write the upwinding as:

$$\mathcal{U}[s^{\text{adv}}](q_L, q_R) = \begin{cases} q_L & \text{if } s^{\text{adv}} > 0 \\ \frac{1}{2}(q_L + q_R) & \text{if } s^{\text{adv}} = 0 \\ q_R & \text{if } s^{\text{adv}} < 0 \end{cases} \tag{9.27}$$

Then the interface states are:

$$\hat{u}_{i+1/2,j} = \mathcal{U}[\hat{u}^{\text{adv}}_{i+1/2,j}](\hat{u}^{n+1/2}_{i+1/2,j,L}, \hat{u}^{n+1/2}_{i+1/2,j,R}) \tag{9.28}$$

$$\hat{v}_{i+1/2,j} = \mathcal{U}[\hat{u}^{\text{adv}}_{i+1/2,j}](\hat{v}^{n+1/2}_{i+1/2,j,L}, \hat{v}^{n+1/2}_{i+1/2,j,R}) \tag{9.29}$$

$$\hat{u}_{i,j+1/2} = \mathcal{U}[\hat{v}^{\text{adv}}_{i,j+1/2}](\hat{u}^{n+1/2}_{i,j+1/2,L}, \hat{u}^{n+1/2}_{i,j+1/2,R}) \tag{9.30}$$

$$\hat{v}_{i,j+1/2} = \mathcal{U}[\hat{v}^{\text{adv}}_{i,j+1/2}](\hat{v}^{n+1/2}_{i,j+1/2,L}, \hat{v}^{n+1/2}_{i,j+1/2,R}) \tag{9.31}$$

Now we can construct the full left and right predictions for the normal velocities on each interface (Eqs. 9.14 and 9.15). This involves simply adding the transverse term to the hat quantities and adding the pressure gradient.

$$u^{n+1/2}_{i+1/2,j,L} = \hat{u}^{n+1/2}_{i+1/2,j,L} - \frac{\Delta t}{2}\left[\frac{1}{2}\left(\hat{v}^{\rm adv}_{i,j-1/2} + \hat{v}^{\rm adv}_{i,j+1/2}\right)\right]\left(\frac{\hat{u}^{n+1/2}_{i,j+1/2} - \hat{u}^{n+1/2}_{i,j-1/2}}{\Delta y}\right) - \frac{\Delta t}{2}(Gp)^{(x),n-1/2}_{i,j} \tag{9.32}$$

and

$$v^{n+1/2}_{i,j+1/2,L} = \hat{v}^{n+1/2}_{i,j+1/2,L} - \frac{\Delta t}{2}\left[\frac{1}{2}\left(\hat{u}^{\rm adv}_{i-1/2,j} + \hat{u}^{\rm adv}_{i+1/2,j}\right)\right]\left(\frac{\hat{v}^{n+1/2}_{i+1/2,j} - \hat{v}^{n+1/2}_{i-1/2,j}}{\Delta x}\right) - \frac{\Delta t}{2}(Gp)^{(y),n-1/2}_{i,j} \tag{9.33}$$

Here $(Gp)^{(x),n-1/2}_{i,j}$ and $(Gp)^{(y),n-1/2}_{i,j}$ are difference-approximations to ∇p in the x and y directions respectively. Note that they are lagged—these come from the projection at the end of the previous timestep. See BCG for a discussion. A similar construction is done for the right states at the interface.

Finally, we do a Riemann solve (again, using the Burger's form of the Riemann problem) followed by upwinding to get the normal advective velocities. This is basically the $\mathcal{R}$ operation followed by $\mathcal{U}$. Together, it is:

$$u^{\rm adv}_{i+1/2,j} = \begin{cases} u^{n+1/2}_{i+1/2,j,L} & \text{if } u^{n+1/2}_{i+1/2,j,L} > 0, \quad u^{n+1/2}_{i+1/2,j,L} + u^{n+1/2}_{i+1/2,j,R} > 0 \\ \frac{1}{2}\left(u^{n+1/2}_{i+1/2,j,L} + u^{n+1/2}_{i+1/2,j,R}\right) & \text{if } u^{n+1/2}_{i+1/2,j,L} \le 0, \quad u^{n+1/2}_{i+1/2,j,R} \ge 0 \\ u^{n+1/2}_{i+1/2,j,R} & \text{otherwise} \end{cases} \tag{9.34}$$

and similar for $v^{\rm adv}_{i,j+1/2}$. These velocities are sometimes referred to as the MAC velocities.

9.3.2 MAC projection

We could simply use these time-centered advective velocities to construct the fluxes through the interfaces and update to the new time level. However BCH showed that such a method is unstable for CFL > 0.5. The fix is to enforce the velocity constraint on these advective velocities. This involves projecting the velocity field onto the space that is divergence free. This projection is usually called the MAC projection. Once the MAC-projected advective velocities are computed, we can reconstruct the interface states using this divergence-free velocity field. Figure 9.2 shows the location of the various quantities that participate it the MAC projection.

The divergence of the MAC velocities is cell-centered and constructed as:

$$(DU)_{i,j} = \frac{u^{\rm adv}_{i+1/2,j} - u^{\rm adv}_{i-1/2,j}}{\Delta x} + \frac{v^{\rm adv}_{i,j+1/2} - v^{\rm adv}_{i,j-1/2}}{\Delta y} \tag{9.35}$$

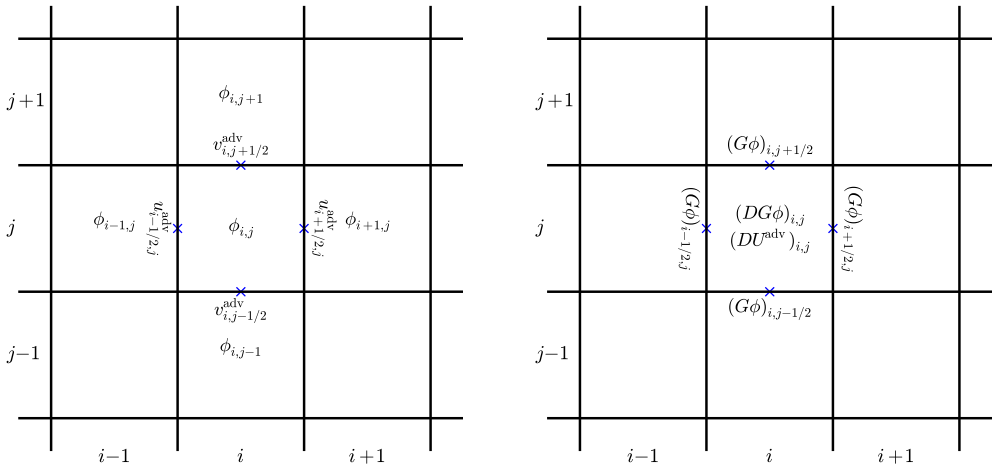

Figure 9.2: The centerings of the various components that make up the MAC projection.

We define a cell-centered ϕ. $G\phi$ will then be edge-centered on a MAC grid, and $L\phi = DG\phi$ is again cell-centered. Since $L = DG$, this makes the MAC projection an exact projection.

We solve

$$L\phi = DU \tag{9.36}$$

using multigrid V-cycles and then update the MAC velocities as:

$$u^{\text{adv}}_{i+1/2,j} = u^{\text{adv}}_{i+1/2,j} - \frac{\phi_{i+1,j} - \phi_{i,j}}{\Delta x} \tag{9.37}$$

$$v^{\text{adv}}_{i,j+1/2} = v^{\text{adv}}_{i,j+1/2} - \frac{\phi_{i,j+1} - \phi_{i,j}}{\Delta y} \tag{9.38}$$

9.3.3 Reconstruct interface states

Next we redo the construction of the interface states. This procedure is identical to that above—construct the interface states $\hat{u}_{L,R}$, $\hat{v}_{L,R}$ on all edges, upwind based on $\hat{u}^{\text{adv}}$, $\hat{v}^{\text{adv}}$, and use these to construct the full states (including transverse terms). Now however, we construct the interface states of u and v on both the x and y-interfaces (not just the normal component at each interface). Finally, instead of solving a Riemann problem to resolve the left and right states, we simply upwind using the MAC-projected u^{adv} and v^{adv}. This results in the interface state $u^{n+1/2}_{i+1/2,j}$, $v^{n+1/2}_{i+1/2,j}$, $u^{n+1/2}_{i,j+1/2}$, $v^{n+1/2}_{i,j+1/2}$.

The only reason we need to do this step over, instead of using the interface states that we predicted previously is we want to ensure that they are consistent with the MAC-projected advective velocities (and therefore, consistent with the constraint).

9.3.4 Provisional update

Once we have the time-centered interface states that are consistent with the MAC-projected advective velocities, we can update the velocities to the new time by discretizing the advective terms $(U \cdot \nabla U)$. We express the advective terms for u as $A_{i,j}^{(u),n+1/2}$ and those for v as $A_{i,j}^{(v),n+1/2}$. These have the form:

$$\begin{aligned} A_{i,j}^{(u),n+1/2} = \quad & \frac{1}{2}\left(u_{i-1/2,j}^{\mathrm{adv}} + u_{i+1/2,j}^{\mathrm{adv}}\right)\frac{u_{i+1/2,j}^{n+1/2} - u_{i-1/2,j}^{n+1/2}}{\Delta x} + \\ & \frac{1}{2}\left(v_{i,j-1/2}^{\mathrm{adv}} + v_{i,j+1/2}^{\mathrm{adv}}\right)\frac{u_{i,j+1/2}^{n+1/2} - u_{i,j-1/2}^{n+1/2}}{\Delta y} \end{aligned} \tag{9.39}$$

$$\begin{aligned} A_{i,j}^{(v),n+1/2} = \quad & \frac{1}{2}\left(u_{i-1/2,j}^{\mathrm{adv}} + u_{i+1/2,j}^{\mathrm{adv}}\right)\frac{v_{i+1/2,j}^{n+1/2} - v_{i-1/2,j}^{n+1/2}}{\Delta x} + \\ & \frac{1}{2}\left(v_{i,j-1/2}^{\mathrm{adv}} + v_{i,j+1/2}^{\mathrm{adv}}\right)\frac{v_{i,j+1/2}^{n+1/2} - v_{i,j-1/2}^{n+1/2}}{\Delta y} \end{aligned} \tag{9.40}$$

The normal update for u, v would include the Gp term and appear as:

$$u_{i,j}^{\star} = u_{i,j}^{n} - \Delta t A_{i,j}^{(u),n+1/2} - \Delta t (Gp)_{i,j}^{(x),n-1/2} \tag{9.41}$$

$$v_{i,j}^{\star} = v_{i,j}^{n} - \Delta t A_{i,j}^{(v),n+1/2} - \Delta t (Gp)_{i,j}^{(y),n-1/2} \tag{9.42}$$

Note that at this point, we don't have an updated p, so we use a lagged value from the previous step's projection.

Alternately, we can note that for an exact projection, Gp, is the gradient of a scalar and would be removed by the projection, so we can omit it in this update, giving an alternate provisional update:

$$u_{i,j}^{\star\star} = u_{i,j}^{n} - \Delta t A_{i,j}^{(u),n+1/2} \tag{9.43}$$

$$v_{i,j}^{\star\star} = v_{i,j}^{n} - \Delta t A_{i,j}^{(v),n+1/2} \tag{9.44}$$

Following the notation in Martin, we distinguish between these with an '$\star$' vs. '$\star\star$' [1].

9.3.5 Approximate projection

This provisional velocity field does not yet obey the constraint. To enforce the constraint, we need to do a projection. Here is where we have the flexibility on

[1] Note that these are identical to ABC [3] approximation projections (1) and (2) (a quick look at ABC might suggest the opposite, but note that their definition of $U^{\star}$ already includes a $-Gp^{n-1/2}$ term, so by explicitly adding it back in, you are dealing with the case where $U^{\star}$ was updated without any $Gp^{n-1/2}$ term, like the '$\star\star$' case above.)

whether to include the $Gp^{n-1/2}$ term. If we were doing an exact projection, then adding the gradient of a scalar would not change the divergence-free velocity field, so there would be no need to add it.

BCH did an exact projection on a cell-centered grid. There, the divergence operator is:

$$(DU)_{i,j} = \frac{u_{i+1,j} - u_{i-1,j}}{2\Delta x} + \frac{v_{i,j+1} - v_{i,j-1}}{2\Delta y} \tag{9.45}$$

This gives a cell-centered DU. If we want ϕ cell-centered, then the gradient, $G\phi$ must also be cell centered so $L\phi = DG\phi$ is cell-centered. This means that we must have

$$(G\phi)^{(x)}_{i,j} = \frac{\phi_{i+1,j} - \phi_{i-1,j}}{2\Delta x} \tag{9.46}$$

$$(G\phi)^{(y)}_{i,j} = \frac{\phi_{i,j+1} - \phi_{i,j+1}}{2\Delta y} \tag{9.47}$$

The resulting Laplacian would then be a 5-point stencil that skips over the immediate neighbors:

$$(L\phi)_{i,j} = \frac{\phi_{i+2,j} - 2\phi_{i,j} + \phi_{i-2,j}}{4\Delta x^2} + \frac{\phi_{i,j+2} - 2\phi_{i,j} + \phi_{i,j-2}}{4\Delta y^2} \tag{9.48}$$

This decomposes the domain into 4 distinct grids that are only linked together at the boundaries. While an exact projection, this decoupling can be undesirable.

Approximate projections relax the idea that $L = DG$. In an exact projection, when you apply the projection operator, P, in succession, the result is unchanged ($P^2 = P$). This is not the case for an approximate projection. As a result, exactly what form you project matters. For an approximate projection, we can use the standard 5-point stencil for the Laplacian,

$$(L\phi)_{i,j} = \frac{\phi_{i+1,j} - 2\phi_{i,j} + \phi_{i-1,j}}{\Delta x^2} + \frac{\phi_{i,j+1} - 2\phi_{i,j} + \phi_{i,j-1}}{\Delta y^2} \tag{9.49}$$

together with the cell-centered divergence above (Eq. 9.45).

Rider [44] and Almgren, Bell, and Crutchfield (ABC) [3] explore various forms of what to project when doing the approximate projection. For instance, do we include the $Gp^{n-1/2}$ term in the provisional velocity or not? Chorin noted that if viscosity is being modeled, then it is necessary to include it here to get second-order accuracy. Also, one can project the update to the velocity, $(U^\star - U^n)/\Delta t$ instead of just the new velocity, since U^n should already be divergence free. Rider argues that projecting the update is not desirable with approximate projections, since any error in U^n being divergence-free is carried forward to the new U^{n+1}. One issue is that a cell-centered approximate projection cannot remove all sources of divergence (see Rider and Martin's PhD thesis).

When projecting the new velocity, we scale by Δt to get a quantity that has dimensions of pressure. The procedure for the projection differs slightly depending on whether we project $U^\star$ or $U^{\star\star}$:

- case I: projecting $U^\star/\Delta t$.

  From the expression above, this looks like:

  $$\frac{U^\star}{\Delta t} = \frac{U^n}{\Delta t} - A^{(u),n+1/2} - (Gp)^{(x),n-1/2} \tag{9.50}$$

  Ideally, U^n is already divergence free, and $Gp^{n-1/2}$ is the gradient of a scalar, which will be removed, so the projection should pick out the divergence free portion of $A^{(u)}$. We solve:

  $$L\phi = D(U^\star/\Delta t) \tag{9.51}$$

  using multigrid V-cycles. We then find the new, divergence free velocity field as:

  $$U^{n+1} = U^\star - \Delta t G\phi \tag{9.52}$$

  Since we already included $Gp^{n-1/2}$ in what we projected, $G\phi$ will be the correction,

  $$G\phi = Gp^{n+1/2} - Gp^{n-1/2} \tag{9.53}$$

  or

  $$Gp^{n+1/2} = Gp^{n-1/2} + G\phi \tag{9.54}$$

  (see Martin 2.5 or ABC). We store this for the next timestep.

- case II: projecting $U^{\star\star}/\Delta t$.

  From the expression above, this looks like:

  $$\frac{U^{\star\star}}{\Delta t} = \frac{U^n}{\Delta t} - A^{(u),n+1/2} \tag{9.55}$$

  There is no explicit $Gp^{n-1/2}$ term. We solve:

  $$L\phi = D(U^{\star\star}/\Delta t) \tag{9.56}$$

  using multigrid V-cycles. We then find the new, divergence free velocity field as:

  $$U^{n+1} = U^{\star\star} - \Delta t G\phi \tag{9.57}$$

  Since there was no $Gp^{n-1/2}$ in what we projected, $p^{n+1/2} = \phi$, and

  $$Gp^{n+1/2} = G\phi \tag{9.58}$$

  We store this for the next timestep.

One pathology of this form of the projection is that $(DU)_{i,j}$ does not actually make use of the velocity field in zone (i, j). This decoupling from the local zone can result in a checkerboarding pattern in the projected velocity field.

9.4 Boundary conditions

For the advection portion of the algorithm, the boundary conditions on u and v are implemented in the usual way, using ghost cells. For the projection,

For a periodic domain, the boundary conditions on ϕ are likewise periodic. At a solid wall or inflow boundary, we already predicted the velocity that we want at the wall (in the advection step), and we do not want this value to change in the projection step. Since the correction is:

$$U^{n+1} = U^\star - \nabla\phi \tag{9.59}$$

we want $\nabla\phi \cdot n = 0$.

At outflow boundaries, we do not want to introduce any shear as we go through the boundary. This means that we do not want any tangential acceleration. Setting $\phi = 0$ on the boundary enforces $\nabla\phi \cdot t = 0$, where t is the unit vector tangential to the boundary.

See ABS for a discussion of the boundary conditions.

9.5 Bootstrapping

At step 0, we do not have a value of $Gp^{-1/2}$. To get an initial value for Gp, we run through the entire evolution algorithm starting with the initial data. At the end of a step, we reset u and v to the initial values and store the Gp at the end of this step as $Gp^{-1/2}$.

It is also common to precede this initialization by first projecting the velocity field to ensure it is divergence free. This way, we do not have to rely on the initial conditions to always set a divergence free velocity field.

9.6 Test problems

9.6.1 Convergence test

Minion [38] introduced a simple test problem with an analytic solution. The velocity field is initialized as:

$$u(x,y) = 1 - 2\cos(2\pi x)\sin(2\pi y) \tag{9.60}$$
$$v(x,y) = 1 + 2\sin(2\pi x)\cos(2\pi y) \tag{9.61}$$

The exact solution at some time t is:

$$u(x,y,t) = 1 - 2\cos(2\pi(x-t))\sin(2\pi(y-t)) \tag{9.62}$$
$$v(x,y,t) = 1 + 2\sin(2\pi(x-t))\cos(2\pi(y-t)) \tag{9.63}$$

Minion also gives the pressure, but this is not needed for the solution. This is run on a doubly-periodic unit square domain. The main utility of this set of initial conditions is that we can use the analytic solution to measure the convergence behavior of the algorithm.

9.7 Extensions

- *Variable density incompressible*: Bell & Marcus [12] describe how to extend these methods to variable density flows. This means that the density in the domain may not be constant, but within a fluid element, the density does not change. This can arise, for instance, in modeling the Rayleigh-Taylor instability.

  The basic idea follows the method described above. Now the mass continuity equation is also evolved:

  $$\frac{\partial \rho}{\partial t} + \nabla \cdot (\rho U) = 0 \tag{9.64}$$

  The density is predicted to the interfaces follow the same procedure above and upwinded using the MAC velocities. For the projection, the decomposition is written as:

  $$U = U^d + \frac{1}{\rho}\nabla\phi \tag{9.65}$$

  and the elliptic equation is now a variable-coefficient equation:

  $$\nabla \cdot \frac{1}{\rho}\nabla\phi = \nabla \cdot U \tag{9.66}$$

- *Viscosity*: When viscosity is included in the system, our momentum equation becomes:

  $$U_t + U \cdot \nabla U + \nabla p = \epsilon \nabla^2 U \tag{9.67}$$

  The solution process for this equation is largely the same. Following BCH, first the advective term is computed by predicting the velocities to the interfaces, doing the MAC projection, and then forming $A^{n+1/2}$. Now there is an explicit viscosity term (at time-level n) present in the prediction, as a source term. The provision velocity update is no longer a simple flux update, but instead requires solving two decoupled diffusion-like equations (one for u and one for v). These are differenced using Crank-Nicolson centering:

  $$\frac{u^\star - u^n}{\Delta t} = -A^{(u),n+1/2} - \nabla p^{(x),n-1/2} + \frac{\epsilon}{2}\nabla^2(u^n + u^\star) \tag{9.68}$$

  $$\frac{v^\star - v^n}{\Delta t} = -A^{(v),n+1/2} - \nabla p^{(y),n-1/2} + \frac{\epsilon}{2}\nabla^2(v^n + v^\star) \tag{9.69}$$

  This involves two separate multigrid solves. Once $U^\star$ is found, the final projection is done as usual.

- *Low Mach number combustion*: In low-Mach number combustion flows, the fluid is allowed to respond to local heat release by expanding. The velocity constraint is derived by differentiating the equation of state along particle paths, leading to the appearance of a source term:

$$\nabla \cdot U = S \tag{9.70}$$

  Here, S, incorporates the compressibility effects due to the heat release and diffusion. This system is used when modeling burning fronts (flames). This type of flow can be thought of as linking two incompressible states (the fuel and the ash) by the expansion across the interface.

  The solution technique largely follows that of the incompressible flow. One caveat, because the constraint now has a local heat source, S, doing the cell-centered divergence described above leads to a decoupling of DU from the local source, since the stencil of DU does not include the zone upon which it is centered.

  This system is described in detail in [42, 25, 11].

- *Stratified flows*: When the background is stratified, a different velocity constraint can arise, capturing the expansion of the fluid element due to the pressure change with altitude. For example, with an ideal gas, the equation of state can be recast as:

$$\nabla \cdot (p_0^{1/\gamma}) = 0 \tag{9.71}$$

  where $p_0(z)$ is the pressure as a function of height, representing the hydrostatic background, and γ is the ratio of specific heats. For a general equation of state, $p_0^(1/\gamma)$ is replaced with something more complicated (see [5, 6, 4] for the development of a low Mach hydrodynamics method for stratified astrophysical flows).

- *Nodal projection*: instead of discretizing the final projection using a cell-centered ϕ, ABS use a node-centered ϕ. While this is still an approximate projection, this discretization couples in the zone we are centered on, and is said to be able to do a better job removing pathological divergent velocity fields that cell-centered projections stumble on.

Chapter 10

Low Mach Number Methods

Incompressible flow represents the zero-Mach number limit of fluid flow—no compressibility effects are modeled. We can extend the ideas of incompressibe flow to allow us to model some compressibility effects, giving rise to low Mach number methods.

10.1 Low Mach divergence constraints

The key idea in solvers for low Mach number flows is that, as a fluid element advects, its pressure remains the same as the background state. For an atmosphere, this background state is a hydrostatic profile. For a smallscale combustion flow, this background state is a spatially constant pressure (and constaint-in-time if the domain is open). We'll denote this background pressure as $p_0(r,t)$, where r is a radial coordinate.

From the first law of thermodynamics, $Tds = de + pd(1/\rho)$, we see

$$T\frac{Ds}{Dt} = \frac{De}{Dt} + p_0\frac{D(1/\rho)}{Dt} \tag{10.1}$$

which is our internal energy evolution equation. The internal energy in a fluid parcel can change due to local heat release and diffusion, so we can write:

$$T\frac{Ds}{Dt} = \frac{De}{Dt} + p_0\frac{D(1/\rho)}{Dt} = H + \nabla\cdot(k\nabla T) \equiv \dot{q} \tag{10.2}$$

where H is the specific energy generation rate (energy / mass /time) and k is the thermal conductivity.

We can derive the constraint on the velocity field by considering the Lagrangian derivative of the pressure—this captures the change in pressure of a fluid element

as it moves through the domain.

$$\frac{Dp_0}{Dt} = \left.\frac{\partial p_0}{\partial \rho}\right|_s \frac{D\rho}{Dt} + \left.\frac{\partial p_0}{\partial s}\right|_\rho \frac{Ds}{Dt} \tag{10.3}$$

we recognize that $\Gamma_1 \equiv \partial \log p_0 / \partial \log \rho|_s$. The Maxwell relations tell us that

$$\partial p_0 / \partial s|_\rho = \rho^2 \partial T / \partial \rho|_s \tag{10.4}$$

This gives us the form:

$$\frac{Dp_0}{Dt} = \frac{p_0}{\rho}\Gamma_1 \frac{D\rho}{Dt} + \rho^2 \left.\frac{\partial T}{\partial \rho}\right|_s \frac{\dot{q}}{T} \tag{10.5}$$

We need an expression for $\partial T / \partial \rho|_s$ and terms of derivatives of ρ and T (since that is what our equations of state typically provide). Consider Γ_1, with $p = p(\rho, T)$:

$$\Gamma_1 = \frac{\rho}{p} \left.\frac{dp}{d\rho}\right|_s = \frac{\rho}{p}\left[p_\rho + p_T \left.\frac{dT}{d\rho}\right|_s\right] \tag{10.6}$$

where we use the shorthand $p_\rho \equiv \partial p / \partial \rho|_T$ and $p_T \equiv \partial p / \partial T|_\rho$. This tells us that

$$\left.\frac{\partial T}{\partial \rho}\right|_s = \frac{p}{\rho p_T}(\Gamma_1 - \chi_\rho) \tag{10.7}$$

where $\chi_\rho \equiv \partial \log p / \partial \log \rho|_T$. Using the following relations

$$\frac{\Gamma_1}{\chi_\rho} = \frac{c_p}{c_v} \tag{10.8}$$

and

$$c_p - c_v = \frac{p}{\rho T}\frac{\chi_T^2}{\chi_\rho} \tag{10.9}$$

with $\chi_T \equiv \partial \log p / \partial \log T|_\rho$ (see, e.g. [29] for both of these relations), we can write

$$\left.\frac{\partial T}{\partial \rho}\right|_s = \frac{p\chi_T}{\rho^2 c_v} \tag{10.10}$$

Putting this all together, we have:

$$\frac{Dp_0}{Dt} = \frac{p_0}{\rho}\Gamma_1 \frac{D\rho}{Dt} + \frac{p\chi_T}{c_v T}\dot{q} \tag{10.11}$$

The continuity equation gives us:

$$\frac{D\rho}{Dt} = -\rho \nabla \cdot U \tag{10.12}$$

Solving for the velocity divergence, we finally have:

$$\begin{aligned}\nabla \cdot U + \frac{1}{\Gamma_1 p_0}\frac{Dp_0}{Dt} &= \frac{\chi_T}{c_v T \Gamma_1}\dot{q} = \frac{\chi_T}{c_p \chi_\rho T}\dot{q} \\ &= \frac{p_T}{\rho c_p p_\rho}\dot{q} \equiv \sigma \dot{q} \end{aligned} \tag{10.13}$$

with

$$\sigma = \frac{\partial p_0/\partial T|_\rho}{\rho c_p \partial p_0/\partial \rho|_T} \tag{10.14}$$

The σ notation follows from [6], where we abbreviate the righthand side as S.

This is the general constraint equation for low Mach flow. Note that the only approximation we made is $p \rightarrow p_0$. This form of the constraint, for a general equation of state, was originally derived in [5].

Combustion limit

A useful limit is smallscale combustion. In an open domain, we can take p_0 as constant, so $Dp_0/Dt = 0$, and we are left with

$$\nabla \cdot U = S \tag{10.15}$$

This looks like the constraint for incompressible flow, but with a source to the divergence. This source captures the compressible effects due to local heat release—as a fluid parcel moves, the only changes to its density will come through local heat release. Methods for this type of constraint are discussed in [42, 25, 11].

Atmospheric case

Another interesting case is that of an atmosphere. If we consider an ideal gas, then $\Gamma_1 = \gamma =$ constant. A second appromation we take is that $p_0 = p_0(r)$—i.e., no time dependence to the atmosphere. Finally, we'll consider the case with no local heat sources ($S = 0$). Then we have

$$\nabla \cdot U + \frac{1}{\gamma p_0} U \cdot \nabla p_0 = 0 \tag{10.16}$$

which is equivalent to

$$\nabla \cdot \left(p_0^{1/\gamma} U \right) = 0 \tag{10.17}$$

This constraint captures the changes in compressibilty due to the background stratification of an atmosphere. This form was originally derived for atmospheric flows by [26] and generalized to stellar flows in [5]. If the structure of the atmosphere is isentropic, then we know that $d \log p_0 = \gamma d \log \rho_0$, where we use ρ_0 to represent the density corresponding to p_0, and we can write this constraint as:

$$\nabla \cdot (\rho_0 U) = 0 \tag{10.18}$$

This is the traditional anelastic constraint.

Extensions involving external heating sources or reactions are discussed in [6, 4], and dealing with the non-constant Γ_1 is discussed in [5, 30]. The time-dependence of the background atmosphere is explored in [6] for plane-parallel atmospheres, following the ideas from [1], and in [40] for spherical, self-gravitating bodies.

10.2 Multigrid for Variable-Density Flows

The solution methodology for these low Mach number systems follows that of the incompressible flow, but with two additions. First, we need to incorporate a density (mass continuity) evolution equation—this will follow the same techniques we already saw for advection, as we'll see later. Next, we need to be able to enforce more general forms of the divergence constraint, which as we'll see in a moment, require us to solve a variable-coefficient elliptic equation. Our multigrid technique will need to be suitably modified.

We now need to solve an elliptic equation of the form:

$$\nabla \cdot (\eta \nabla \phi) = f \tag{10.19}$$

If we denote the discrete divergence and gradient operators as D and G, then our operator will be $L_\eta \equiv D\eta G$. If we wish to use a cell-centered discretization for ϕ, then using a standard centered-difference for D and G will result in a stencil that reaches two zones on either side of the current zone. This can lead to an odd-even decoupling.

he appropriate
paper
is a Pao and
a (or Pao's
?) that also
sses issues with
entered

Instead, we again use an approximate projection. We discretize the variable-coefficient Laplacian as:

$$(L_\eta \phi)_{i,j} = \frac{\eta_{i+1/2,j}(\phi_{i+1,j} - \phi_{i,j}) - \eta_{i-1/2,j}(\phi_{i,j} - \phi_{i-1,j})}{\Delta x^2} + \frac{\eta_{i,j+1/2}(\phi_{i,j+1} - \phi_{i,j}) - \eta_{i,j-1/2}(\phi_{i,j} - \phi_{i,j-1})}{\Delta y^2} \tag{10.20}$$

We can define the interface values of η as the averages of the cell-centered values, e.g.,

$$\eta_{i,j+1/2} = \frac{1}{2}(\eta_{i,j} + \eta_{i,j+1}) \tag{10.21}$$

Our elliptic equation is then

$$(L_\eta \phi)_{i,j} = f_{i,j} \tag{10.22}$$

The relaxation method for this operator again relies on isolating $\phi_{i,j}$, yielding:

$$\phi_{i,j} = \frac{\tilde{\eta}_{i+1/2,j}\phi_{i+1,j} + \tilde{\eta}_{i-1/2,j}\phi_{i-1,j} + \tilde{\eta}_{i,j+1/2}\phi_{i,j+1} + \tilde{\eta}_{i,j-1/2}\phi_{i,j-1} - f_{i,j}}{\tilde{\eta}_{i+1/2,j} + \tilde{\eta}_{i-1/2,j} + \tilde{\eta}_{i,j+1/2} + \tilde{\eta}_{i,j-1/2}} \tag{10.23}$$

with the shorthand that $\tilde{\eta}_{i\pm1/2,j} = \eta_{i\pm1/2,j}/\Delta x^2$ and $\tilde{\eta}_{i,j\pm1/2} = \eta_{i,j\pm1/2}/\Delta y^2$.

To put this into our multigrid framework, there are three changes we need to make:

- The smoothing function needs to implement the more general smoothing described by Eq. 10.23.
- The residual function needs to compute $(L_\eta \phi)_{i,j}$ according to Eq. 10.20, and then $r_{i,j} = f_{i,j} - (L_\eta \phi)_{i,j}$.
- The coefficients, η should be averaged to the edges on the fine grid and then restricted down the multigrid hierarchy as edge-based quantities.

10.2.1 Test problem

Periodic

To test the solver, we need to devise a problem with a known analytic solution. The easiest way to do this is to pick an $\eta(x)$ and ϕ and then do the divergence and gradients to find the required righthand side, f. We'll use periodic BCs, and for our equation $\nabla \cdot (\eta \nabla \phi) = f$, the following provide a well-posed test problem:

$$\begin{aligned} \eta &= 2 + \cos(2\pi x)\cos(2\pi y) \\ f &= -16.0\pi^2 \left[\cos(2\pi x)\cos(2\pi y) + 1\right] \sin(2\pi x)\sin(2\pi y) \end{aligned} \tag{10.24}$$

with the solution:

$$\phi^{\text{true}} = \sin(2\pi x)\sin(2\pi y)$$

There is an important caveat when dealing with a purely-periodic problem. Since there is no boundary values to "anchor" the solution, it is free to float. Solving the elliptic problem will give use the correct $\nabla\phi$, but the average of ϕ over the domain is unconstrained. For our algorithms, it is $\nabla\phi$ that matters (that is the forcing term that enters into the momentum equation).

When for checking convergence, we want to compare to the exact solution. We therefore normalize ϕ by subtracting off its average value before computing the norm of the error with respect to the exact solution:

$$\epsilon = \| \phi_{i,j} - \bar{\phi} - \phi^{\text{true}}_{i,j} \| \ , \tag{10.25}$$

where

$$\bar{\phi} = \frac{1}{N_x N_y} \sum_{i,j} \phi_{i,j} \tag{10.26}$$

As discussed in § 6.5, this can arise if the discrete form the righthand side, $f_{i,j}$ does not sum exactly to zero. Figure 10.1 shows the solution to this problem with a 512^2 grid. and the convergence of the solver described here is shown in Figure 10.2.

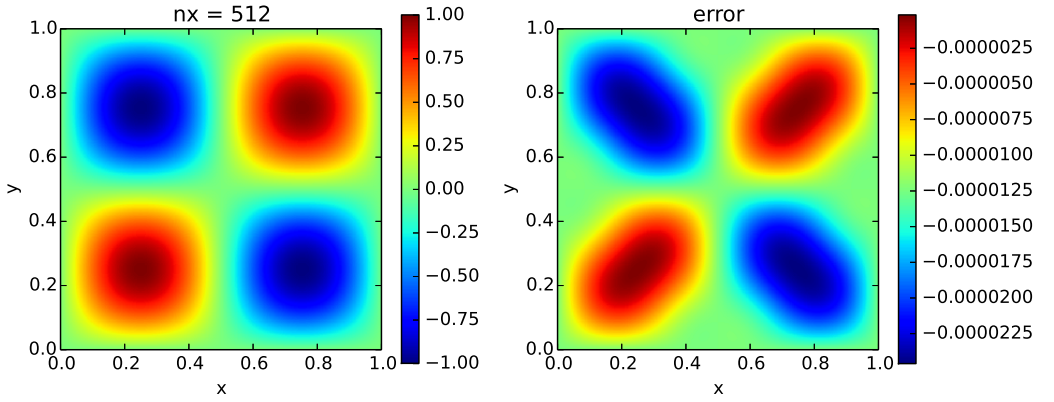

Figure 10.1: Solution and error to the variable-coefficient Poisson problem defined in Eq. 10.24. This test can be run with pyro `multigrid/test_mg_vc_periodic.py`.

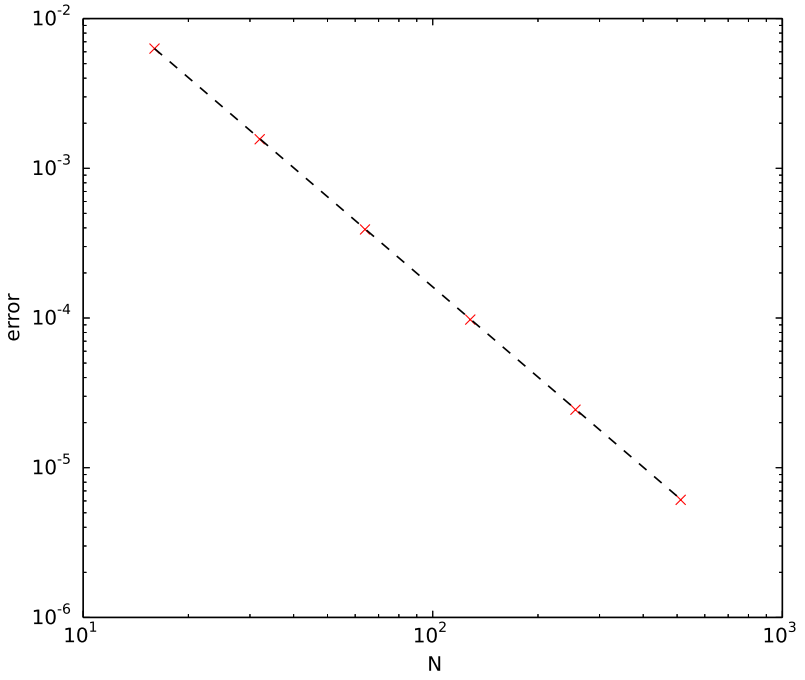

Figure 10.2: Convergence of the variable-coefficient multigrid solver for the test problem defined in Eq. 10.24. This test can be run with pyro `multigrid/test_mg_vc_periodic.py`.

Dirichlet

We can run the same problem with Dirichlet boundary conditions on ϕ, and we are free to pick different boundary conditions for η, since it represents a different physical quantity. Since we only have homogeneous Dirichlet or Neumann BCs implemented, we'll run with Neumann BCs on η.

10.3 Atmospheric flows

10.3.1 Equation Set

For atmospheric flows, we define a one-dimensional base state that is in hydrostatic equilibrium. In general, this base state can be time-dependent, expanding in response to heating in the atmosphere (see e.g. [1, 6]. Here we'll consider only a time-independent state.

We'll follow the procedure defined in [40]: we define ρ_0 as the lateral average of ρ:

$$\rho_{0_j} = \frac{1}{N_x} \sum_i \rho_{i,j} \tag{10.27}$$

and then we define the base state pressure, p_0, by integrating the equation of hydrostatic equilibrium, $dp_0/dy = \rho g$, as:

$$p_{0_{j+1}} = p_{0_j} + \frac{\Delta y}{2}(\rho_{0_j} + \rho_{0_{j+1}})g \tag{10.28}$$

with an initial condition of

$$p_{0_{\text{jlo}}} = \frac{1}{N_x} \sum_i p_{i,\text{jlo}}^{\text{initial}} \tag{10.29}$$

The compressible momentum equation (written in terms of velocity is):

$$\rho \frac{\partial U}{\partial t} + \rho U \cdot \nabla U + \nabla p = \rho g \tag{10.30}$$

Subtracting off the base state, and defining the perturbational pressure (sometimes called the dynamic pressure) as $p' = p - p_0$, and perturbational density as $\rho' = \rho - \rho_0$, we have:

$$\rho \frac{\partial U}{\partial t} + \rho U \cdot \nabla U + \nabla p' = \rho' g \tag{10.31}$$

or

$$\frac{\partial U}{\partial t} + U \cdot \nabla U + \frac{1}{\rho} \nabla p' = \frac{\rho'}{\rho} g \tag{10.32}$$

Several authors [30, 52] explored the idea of energy conservation in a low Mach number system and found that an additional term (which can look like a buoyancy) is needed in the low Mach number formulation, yielding:

$$\frac{\partial U}{\partial t} + U \cdot \nabla U + \frac{\beta_0}{\rho} \nabla \left(\frac{p'}{\beta_0} \right) = \frac{\rho'}{\rho} g \tag{10.33}$$

Completing the system are the continuity equation,

$$\frac{\partial \rho}{\partial t} + \nabla \cdot (\rho U) = 0 \tag{10.34}$$

and the constraint,

$$\nabla \cdot (\beta_0 U) = 0 \tag{10.35}$$

with $\beta_0 = p_0^{1/\gamma}$.

10.3.2 Solution Procedure

The general solution procedure is for a single step is:

I. *Predict U to the interfaces*

II. *Enforce the divergence constraint on the interface U's (the MAC projection) to get* $U^{\rm adv}$.

Decomposing the velocity field as

$$U^\star = U^d + \frac{\beta_0}{\rho}\nabla\phi \tag{10.36}$$

as suggested from the form of the momentum equation, our Poisson equation can be defined by multiplying by β_0 and taking the divergence, and using $\nabla \cdot (\beta_0 U^d) = 0$, giving

$$\nabla \cdot \frac{\beta_0^2}{\rho}\nabla\phi = \nabla \cdot (\beta_0 U^\star) \tag{10.37}$$

is a Δt here

Note that when written like this, ϕ has units of p'/β_0.

For the MAC projection, we have edge-centered velocities (in their respective coordinate direction). We define an edge-centered β_0 as

$$\beta_{0_{j+1/2}} = \frac{1}{2}(\beta_{0_j} + \beta_{0_{j+1}}) \tag{10.38}$$

(note that, since β_0 is one-dimensional, we only average in the vertical direction). The divergence term is then:

$$[\nabla \cdot (\beta_0 U)]^{\rm adv}_{i,j} = \beta_{0_j}\frac{u^{\rm adv}_{i+1/2,j} - u^{\rm adv}_{i-1/2,j}}{\Delta x} + \frac{\beta_{0_{j+1/2}} v^{\rm adv}_{i,j+1/2} - \beta_{0_{j-1/2}} v^{\rm adv}_{i,j-1/2}}{\Delta y} \tag{10.39}$$

We define the coefficient, η, in the Poisson equation as:

$$\eta_{i,j} = \frac{{\beta_0}_j^2}{\rho_{i,j}} \tag{10.40}$$

and bring it to edges simply through averaging. Multigrid is used to solve for $\phi_{i,j}$.

We then solve

$$(L_\eta \phi)_{i,j} = [\nabla \cdot (\beta_0 U)]^{\rm adv}_{i,j} \tag{10.41}$$

Once we solve for ϕ, we correct the velocity as:

$$U^{\text{new}} = U^{\star} - \frac{\beta_0}{\rho}\nabla\phi \tag{10.42}$$

Since the MAC velocities are edge-centered, our correction appears as:

$$u^{\text{adv}}_{i+1/2,j} = u^{\text{adv}}_{i+1/2,j} - \left(\frac{\beta_0}{\rho}\right)_{i+1/2,j} \frac{\phi_{i+1,j} - \phi_{i,j}}{\Delta x} \tag{10.43}$$

$$v^{\text{adv}}_{i,j+1/2} = v^{\text{adv}}_{i,j+1/2} - \left(\frac{\beta_0}{\rho}\right)_{i,j+1/2} \frac{\phi_{i,j+1} - \phi_{i,j}}{\Delta y} \tag{10.44}$$

III. *Predict ρ to the interfaces*

We need to solve the continuity equation. We use the same techniques that were used for advection. Our equation is:

$$\rho_t + (\rho u)_x + (\rho v)_y = 0 \tag{10.45}$$

The x-interface left state would be:

$$\begin{aligned}
\rho^{n+1/2}_{i+1/2,j,L} &= \rho^n_{i,j} + \frac{\Delta x}{2}\frac{\partial \rho}{\partial x} + \frac{\Delta t}{2}\frac{\partial \rho}{\partial t} + \dots \\
&= \rho^n_{i,j} + \frac{\Delta x}{2}\frac{\partial \rho}{\partial x} + \frac{\Delta t}{2}\left[-(\rho u)_x - (\rho v)_y\right]_{i,j} \\
&= \underbrace{\rho^n_{i,j} + \frac{\Delta x}{2}\left(1 - \frac{\Delta t}{\Delta x}u_{i,j}\right)\frac{\partial \rho}{\partial x}}_{\hat{\rho}^{n+1/2}_{i+1/2,j,L}} - \frac{\Delta t}{2}\left[\rho u_x\right]_{i,j} - \frac{\Delta t}{2}\left[(\rho v)_y\right]_{i,j}
\end{aligned} \tag{10.46}$$

A similar construction would yield the right state at that interface, and the y-interface states.

Since we already have the MAC-projected advected velocities at this point, we can use them in the construction of the interface states and the upwinding. As before, we split this into a normal part (the $\hat{\rho}$ term) and "transverse" part (which here includes the non-advective part of the divergence). We first construct the normal parts as

$$\hat{\rho}^{n+1/2}_{i+1/2,j,L} = \rho^n_{i,j} + \frac{1}{2}\left(1 - \frac{\Delta t}{\Delta x}u^{\text{adv}}_{i+1/2,j}\right)\overline{\Delta\rho}^{(x)}_{i,j} \tag{10.47}$$

and then solve the Riemann problem (upwinding) for these:

$$\hat{\rho}^{n+1/2}_{i+1/2,j} = \mathcal{U}[u^{\text{adv}}_{i+1/2,j}](\hat{\rho}^{n+1/2}_{i+1/2,j,L}, \hat{\rho}^{n+1/2}_{i+1/2,j,R}) = \begin{cases} \hat{\rho}^{n+1/2}_{i+1/2,j,L} & u^{\text{adv}}_{i+1/2,j} > 0 \\ \hat{\rho}^{n+1/2}_{i+1/2,j,R} & u^{\text{adv}}_{i+1/2,j} < 0 \end{cases} \tag{10.48}$$

The same procedure is done for the y-interfaces.

The full states are then constructed using these "hat" states. For example,

$$\rho^{n+1/2}_{i+1/2,j,L} = \hat{\rho}^{n+1/2}_{i+1/2,j,L} - \frac{\Delta t}{2}\rho_{i,j}\frac{u^{\rm adv}_{i+1/2,j} - u^{\rm adv}_{i-1/2,j}}{\Delta x} - \frac{\Delta t}{2}\frac{\hat{\rho}^{n+1/2}_{i,j+1/2}v^{\rm adv}_{i,j+1/2} - \hat{\rho}^{n+1/2}_{i,j-1/2}v^{\rm adv}_{i,j-1/2}}{\Delta y} \tag{10.49}$$

Once the new states on both the x- and y-interfaces are constructed, we again upwind to find the final ρ state on each interface:

$$\rho^{n+1/2}_{i+1/2,j} = \mathcal{U}[u^{\rm adv}_{i+1/2,j}](\rho^{n+1/2}_{i+1/2,j,L}, \rho^{n+1/2}_{i+1/2,j,R}) = \begin{cases} \rho^{n+1/2}_{i+1/2,j,L} & u^{\rm adv}_{i+1/2,j} > 0 \\ \rho^{n+1/2}_{i+1/2,j,R} & u^{\rm adv}_{i+1/2,j} < 0 \end{cases} \tag{10.50}$$

IV. *Do the conservative update of ρ*

Once the interface states are found, we can conservatively-difference the continuity equation:

$$\rho^{n+1}_{i,j} = \rho^{n}_{i,j} - \frac{\Delta t}{\Delta x}\left(\rho^{n+1/2}_{i+1/2,j}u^{\rm adv}_{i+1/2,j} - \rho^{n+1/2}_{i-1/2,j}u^{\rm adv}_{i-1/2,j}\right) - \frac{\Delta t}{\Delta y}\left(\rho^{n+1/2}_{i,j+1/2}v^{\rm adv}_{i,j+1/2} - \rho^{n+1/2}_{i,j-1/2}v^{\rm adv}_{i,j-1/2}\right) \tag{10.51}$$

V. *Update U^n to $U^{n+1,\star}$*

VI. *Enforce the divergence constraint on U^{n+1}*

For the final projection, we have cell-centered velocties. We define the divergence term as:

$$[\nabla\cdot(\beta_0 U)]^{\star}_{i,j} = \beta_{0_j}\frac{u^{\star}_{i+1,j} - u^{\star}_{i-1,j}}{2\Delta x} + \frac{\beta_{0_{j+1}}v^{\star}_{i,j+1} - \beta_{0_{j-1}}v^{\star}_{i,j-1}}{2\Delta y} \tag{10.52}$$

10.3.3 Timestep constraint

In addition to the advective timestep constraint, an additional constraint is needed if the velocity field is initialially zero. In [4], a constraint based on the buoyancy forcing is used. First, we compute the maxiumum buoyancy acceleration,

$$a_{\rm max} = \max\left\{\left|\frac{\rho' g}{\rho}\right|\right\} \tag{10.53}$$

and then

$$\Delta t_{\rm force} = \left(\frac{2\Delta x}{a_{\rm max}}\right)^{1/2} \tag{10.54}$$

This is based simply on $\Delta x = (1/2)at^2$—the time it takes for the buoyant force to accelerate a fluid element across a zone width.

10.3.4 Bootstrapping

First we need to make sure that the initial velocity field satisfies our constraint

Next we need to find $\nabla \pi^{n-1/2}$ for the first step.

10.4 Combustion

Taking $p = p_0 + \pi$, with $p_0 =$ constant, the system becomes:

$$\frac{\partial \rho}{\partial t} + \nabla \cdot (\rho U) = 0 \tag{10.55}$$

$$\frac{\partial \rho U}{\partial t} + \nabla \cdot (\rho U U) + \nabla \pi = 0 \tag{10.56}$$

$$\nabla \cdot U = S \tag{10.57}$$

10.4.1 Species

10.4.2 Constraint

Our constraint equation is $\nabla \cdot U = S$. Decomposing the velocity field as

$$U^\star = U^d + \frac{1}{\rho} \nabla \phi \tag{10.58}$$

our Poisson equation can be defined by taking the divergence, and using $\nabla \cdot U^d = S$, giving

$$\nabla \cdot \frac{1}{\rho} \nabla \phi = \nabla \cdot U^\star - S \tag{10.59}$$

10.4.3 Solution Procedure

The general solution procedure is for a single step is:

- React for $\Delta t/2$
- Do the hydrodynamcis for Δt
  - Predict U to the interfaces
  - Enforce the divergence constraint on the interface U's (the MAC projection) to get U^{adv}.
  - Predict ρ to the interfaces
  - Do the conservative update of ρ

  - Update U^n to U^{n+1}
  - Enforce the divergence constraint on U^{n+1}
- React for $\Delta t/2$

Chapter 11

Radiation Hydrodynamics

11.1 Equations of Radiation Hydrodynamics

11.1.1 Reference Frames

11.1.2 Angular Approximations / Moments

11.1.3 Closures

11.2 Hyperbolic System

11.3 Parabolic System

11.3.1 General Elliptic Solver

For the gray radiation hydrodynamics system, we need to solve a linear system that takes the form:

$$\alpha\phi + \nabla \cdot \beta\nabla\phi + \gamma \cdot \nabla\phi = f \tag{11.1}$$

(see Zhang et al., Eq. 45). We can discretize this for a cell-centered ϕ to second-order as:

$$\begin{aligned}\alpha_{i,j}\phi_{i,j} + \frac{(\beta\nabla\phi)_{i+1/2,j} - (\beta\nabla\phi)_{i-1/2,j}}{\Delta x} + \frac{(\beta\nabla\phi)_{i,j+1/2} - (\beta\nabla\phi)_{i,j-1/2}}{\Delta y} \\ + \gamma^{(x)}_{i,j}\frac{\phi_{i+1,j} - \phi_{i-1,j}}{2\Delta x} + \gamma^{(y)}_{i,j}\frac{\phi_{i,j+1} - \phi_{i,j-1}}{2\Delta y} = f_{i,j}\end{aligned} \tag{11.2}$$

where we decompose the vector, γ as $\gamma = \gamma^{(x)}\hat{x} + \gamma^{(y)}\hat{y}$. Expanding the gradients:

$$\begin{aligned}\alpha_{i,j}\phi_{i,j} &+ \frac{\beta_{i+1/2,j}(\phi_{i+1,j} - \phi_{i,j}) - \beta_{i-1/2,j}(\phi_{i,j} - \phi_{i-1,j})}{\Delta x^2} \\ &+ \frac{\beta_{i,j+1/2}(\phi_{i,j+1} - \phi_{i,j}) - \beta_{i,j-1/2}(\phi_{i,j} - \phi_{i,j-1})}{\Delta y^2} \\ &+ \gamma^{(x)}_{i,j}\frac{\phi_{i+1,j} - \phi_{i-1,j}}{2\Delta x} + \gamma^{(y)}_{i,j}\frac{\phi_{i,j+1} - \phi_{i,j-1}}{2\Delta y} = f_{i,j} \end{aligned} \tag{11.3}$$

There are several different linear system algorithms people use to solve these types of systems. Since we have already developed a multigrid solver, we will add equations of this type to the multigrid framework.

Defining $\tilde{\beta}_{i\pm1/2,j} \equiv \beta_{i\pm1/2,j}/\Delta x^2$, $\tilde{\beta}_{i,j\pm1/2} \equiv \beta_{i,j\pm1/2}/\Delta y^2$, $\tilde{\gamma}^{(x)}_{i,j} = \gamma^{(x)}_{i,j}/(2\Delta x)$, and $\tilde{\gamma}^{(y)}_{i,j} = \gamma^{(y)}_{i,j}/(2\Delta y)$, we have as an update to $\phi_{i,j}$,

$$\begin{aligned}\phi_{i,j} = \frac{1}{D_{i,j}}\Big[f_{i,j} &- (\tilde{\beta}_{i+1/2,j} + \tilde{\gamma}^{(x)}_{i,j})\phi_{i+1,j} - (\tilde{\beta}_{i-1/2,j} - \tilde{\gamma}^{(x)}_{i,j})\phi_{i-1,j} \\ &- (\tilde{\beta}_{i,j+1/2} + \tilde{\gamma}^{(y)}_{i,j})\phi_{i,j+1} - (\tilde{\beta}_{i,j-1/2} - \tilde{\gamma}^{(y)}_{i,j})\phi_{i,j-1}\Big] \end{aligned} \tag{11.4}$$

with

$$D_{i,j} = \alpha_{i,j} - \tilde{\beta}_{i+1/2,j} - \tilde{\beta}_{i-1/2,j} - \tilde{\beta}_{i,j+1/2} - \tilde{\beta}_{i,j-1/2} \tag{11.5}$$

The remaining details of the multigrid solver are unchanged. The boundary conditions are implemented in the same way as done for Poisson's equation. For radiation, we will need to implement inhomogeneous boundary conditions, which we can do via the ghost cell filling using Eqs. 6.27 and 6.31. Note that because of this general form of the equation, it is no longer simple to do the boundary charge method described in § 6.5, so we must modify the ghost cell fill routines explicitly.

The only changes to the core solver are in the smoothing function, which implements Eq. 11.4 and in the residual, which constructs our operator as discretized in Eq. 11.3.

To test this solver out, we can define a test problem by picking functional forms of α, β, and γ, and a solution, ϕ, with desired boundary conditions and find the resulting f matching:

$$\alpha\phi + \nabla\cdot(\beta\nabla\phi) + \gamma\cdot\nabla\phi = f \tag{11.6}$$

Let's find a set of coefficients and righthand side for which

$$\phi = \cos(\pi x/2)\cos(\pi y/2) \tag{11.7}$$

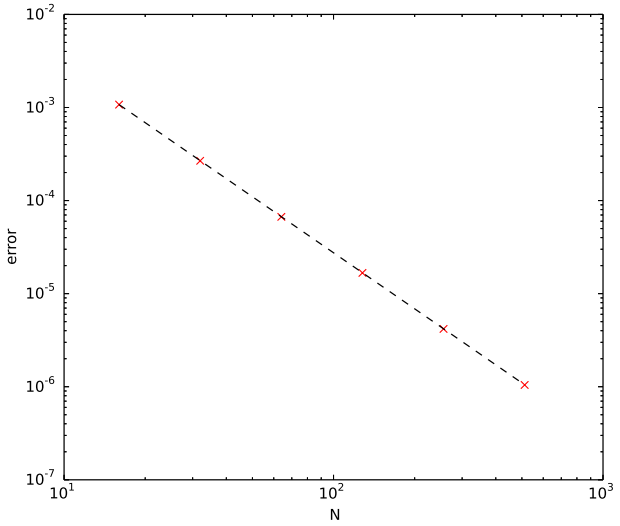

Figure 11.1: Convergence of the multigrid solver on our test problem $\alpha\phi + \nabla \cdot (\beta\nabla\phi) + \gamma \cdot \nabla\phi = f$ with inhomogeneous Dirichlet boundary conditions.

is a solution on $[0,1] \times [0,1]$. This would satisfy the Dirichlet boundary conditions:

$$\phi|_{x=0} = \cos(\pi y/2) \tag{11.8}$$
$$\phi|_{x=1} = 0 \tag{11.9}$$
$$\phi|_{y=0} = \cos(\pi x/2) \tag{11.10}$$
$$\phi|_{y=1} = 0 \tag{11.11}$$

For the coefficients, we choose: on with homogeneous Dirichlet boundary conditions. We use

$$\alpha = 10 \tag{11.12}$$
$$\beta = xy + 1 \tag{11.13}$$
$$\gamma = \hat{x} + \hat{y} \tag{11.14}$$

This gives

$$f = -\frac{\pi}{2}(x+1)\sin\left(\frac{\pi y}{2}\right)\cos\left(\frac{\pi x}{2}\right) - \frac{\pi}{2}(y+1)\sin\left(\frac{\pi x}{2}\right)\cos\left(\frac{\pi y}{2}\right) + \left(10 - \frac{xy+1}{2}\pi^2\right)\cos\left(\frac{\pi x}{2}\right)\cos\left(\frac{\pi y}{2}\right) \tag{11.15}$$

Figure 11.1 shows the convergence of multigrid for this problem, and figure 11.2 shows the solution. We see second order convergence, as expected with this discretization.

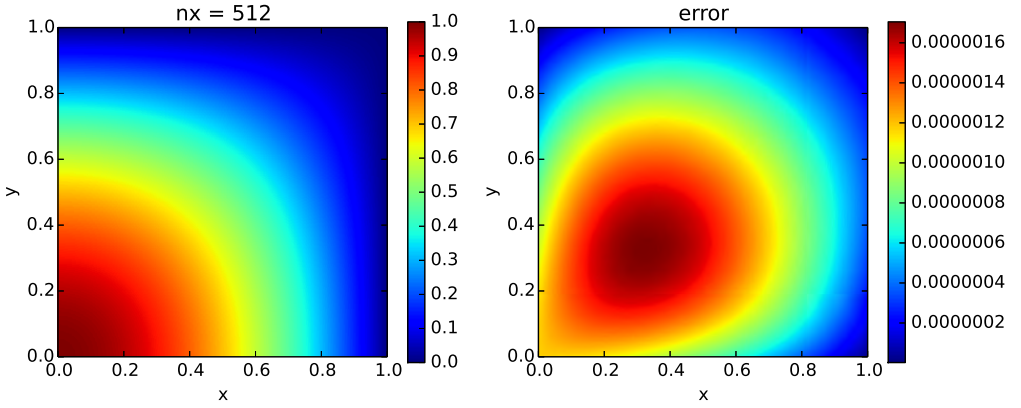

Figure 11.2: The solution to our general elliptic text problem, Eqs. 11.6 to 11.15 with inhomogeneous boundary conditions on a 512^2 grid. This test can be run in pyro `multigrid/test_mg_general_inhomogeneous.py`.

Appendix A

Using hydro_examples

Here we describe the basic structure and use of the hydro_examples *codes that implement standalone, simple 1-d versions of the algorithms described in these notes.*

The hydro_examples codes are simple 1-d solvers written in python that illustrate many of the ideas in these nodes. They are used for making many of the figures found throughout (wherever the "  hydro_examples: " note is found).

A.1 Getting hydro_examples

The hydro_examples codes are hosted on github:
`https://github.com/zingale/hydro_examples`

There are a few ways to get them. The simplest way is to just clone from github on the commandline:

```
git clone https://github.com/zingale/hydro_examples
```

This will create a local git repo on your machine with all of the code. You can then run the scripts, "as is" without needing to go to github anymore. Periodically, there may be updates to the scripts, which you can obtain by issuing "`git pull`" in the `hydro_examples/` directory. Note that if you have made any local changes to the scripts, `git` may complain. The process of merging changes is descibed online in various places, e.g. "Resolving a merge conflict from the command line" from github.

Alternately, you can use github's web interface to fork it. Logon to github (or create an account) and click on the "Fork" icon on the hydro_examples page. You can then interact with your version of the repo as needed. This method will allow

you to push changes back to github, and, if you think they should be included in the main hydro_examples repo, issue a pull-request.

A.2 hydro_examples codes

The codes in hydro_examples are organized into directories named after the chapters in these notes. Each directory is self-contained. The following scripts are available:

- `advection/`
  - `advection.py`: a 1-d second-order linear advection solver with a wide range of limiters.
  - `fdadvect_implicit.py`: a 1-d first-order implicit finite-difference advection solver using periodic boundary conditions.
  - `fdadvect.py`: a 1-d first-order explicit finite-difference linear advection solver using upwinded differencing.
- `burgers/`
  - `burgers.py`: a 1-d second-order solver for the inviscid Burgers' equation, with initial conditions corresponding to a shock and a rarefaction.
- `compressible/`
  - `euler.ipynb`: an IPython notebook using SymPy that derives the eigensystem for the primitive-variable form of the Euler equations.
  - `riemann-phase.py`: draw the Hugoniot curves in the u-p plane for a pair of states that comprise a Riemann problem.
- `finite-volume/`
  - `conservative-interpolation.ipynb`: an IPython notebook using SymPy that derives conservative interpolants.
- `multigrid/`
  - `mg_converge.py`: a convergence test of the multigrid solver. A Poisson problem is solved at various resolutions and compared to the exact solution. This demonstrates second-order accuracy.
  - `mg_test.py`: a simple driver for the multigrid solver. This sets up and solves a Poisson problem and plots the behavior of the solution as a function of V-cycle number.
  - `multigrid.py`: a multigrid class for cell-centered data. This implements pure V-cycles. A square domain with 2^N zones (N a positive integer) is required.

  - `patch1d.py`: a class for 1-d cell-centered data that lives on a grid. This manages the data, handles boundary conditions, and provides routines for prolongation and restriction to other grids.

- `diffusion/`

  - `diffusion-explicit.py`: solve the constant-diffusivity diffusion equation explicitly. The method is first-order accurate in time, but second-order in space. A Gaussian profile is diffused—the analytic solution is also a Gaussian.

  - `diffusion-implicit.py`: solve the constant-diffusivity diffusion equation implicitly. Crank-Nicolson time-discretization is used, resulting in a second-order method. A Gaussian profile is diffused.

- `multiphysics/`

  - `burgersvisc.py`: solve the viscous Burgers equation. The advective terms are treated explicitly with a second-order accurate method. The diffusive term is solved using an implicit Crank-Nicolson discretization. The overall coupling is second-order accurate.

  - `diffusion-reaction.py`: solve a diffusion-reaction equation that propagates a diffusive reacting front (flame). A simple reaction term is modeled. The diffusion is solved using a second-order Crank-Nicolson discretization. The reactions are evolved using the VODE ODE solver (via SciPy). The two processes are coupled together using Strang-splitting to be second-order accurate in time.

Appendix B

Using pyro

Here we describe the basic structure and use of the pyro code that implements many of the algorithms described in these notes.

B.1 Getting pyro

pyro can be downloaded from its github repository, `https://github.com/zingale/pyro2` as:

```
git clone https://github.com/zingale/pyro2
```

The structure of the code and descriptions of the various runtime parameters is found on the pyro webpage, `http://zingale.github.io/pyro2/`, and described in [55].

pyro uses the matplotlib and numpy libraries. Several routines are written in Fortran and need to be compiled. The script `mk.sh` will build the packages. This requires that `f2py` is installed.

B.2 The pyro Solvers

pyro offers the following 2-d solvers:

- *advection*: an unsplit, second-order method for linear advection, following the ideas from Chapter 3.
- *compressible*: an unsplit, second-order compressible hydrodynamics solver using the piecewise linear reconstruction discussed in Chapter 5.
- *diffusion*: a second-order implicit diffusion solver, based on the ideas from Chapter 7.

- *incompressible*: a second-order incompressible hydrodynamics solver using a cell-centered approximate projection, as discussed in Chapter 9.
- *lm_atm*: a low-Mach number hydrodynamics solver for atmospheric flows, as discussed in S 10.3
- *multigrid*: a multigrid solver for constant-coefficient Helmholtz elliptic equations. This follows the ideas from Chapter 6, and is used by the diffusion and incompressible solvers.

B.3 pyro's Structure

The grid structure is managed by the `patch.Grid2d` class. Data is that lives on the grid is contained in a `patch.CellCenterData2d` object. Methods are available to provide access to the data and fill the ghost cells.

Each pyro solver is its own python module. All but the multigrid solver represent time-dependent problems. Each of these provide a `Simulation` class that provides the routines necessary to initialize the solver, determine the timestep, and evolve the solution in time. Each solver has one or more sets of initial conditions defined in the solver's `problems/` directory.

All time-dependent problems are run through the `pyro.py` script. The general form is:

`./pyro.py` *solver problem inputs*

where *solver* is one of `advection`, `compressible`, `diffusion`, `incompressible`, *problem* is one of the problems defined in the solver's `problems/` directory, and *inputs* is the name of an input file that defines the values of runtime parameter.

The possible runtime parameters and their defaults are defined in the `_defaults` files in the main directory and each solver and problem directory. Note that the inputs file need not be in the `pyro2/` directory. The solver's `problems/` directory will also be checked.

B.4 Running pyro

A simple Gaussian advection simulation is provided by the advection *smooth* problem. This is run as:

```
./pyro.py advection smooth inputs.smooth
```

As this is run, the solution will be visualized at each step, showing the progression of the simulation.

solver	problem	problem description
advection	smooth	advect a smooth Gaussian profile
	tophat	advect a discontinuous tophat profile
compressible	bubble	a buoyant bubble in a stratified atmosphere
	kh	setup a shear layer to drive Kelvin-Helmholtz instabilities
	quad	2-d Riemann problem based on [46]
	rt	a simple Rayleigh-Taylor instability
	sedov	the classic Sedov-Taylor blast wave
	sod	the Sod shock tube problem
diffusion	gaussian	diffuse an initial Gaussian profile
incompressible	converge	A simple incompressible problem with known analytic solution.
	shear	a doubly-periodic shear layer
lm_atm	bubble	a buoyant bubble in a stratified atmosphere

Table B.1: Solvers and their distributed problems

A list of the problems available for each solver is given in Table B.1. For the multigrid solver, there are scripts available in the `multigrid/` directory that illustrate its use.

Bibliography

[1] A. Almgren. A new look at the pseudo-incompressible solution to Lamb's problem of hydrostatic adjustment. 57:995–998, April 2000. (Cited on pages 136 and 139)

[2] A. S. Almgren, V. E. Beckner, J. B. Bell, M. S. Day, L. H. Howell, C. C. Joggerst, M. J. Lijewski, A. Nonaka, M. Singer, and M. Zingale. CASTRO: A New Compressible Astrophysical Solver. I. Hydrodynamics and Self-gravity. *Astrophys J*, 715:1221–1238, June 2010. (Cited on pages 65 and 83)

[3] A. S. Almgren, J. B. Bell, and W. Y. Crutchfield. Approximate projection methods: Part I. Inviscid analysis. *SIAM J. Sci. Comput.*, 22(4):1139–59, 2000. (Cited on pages 127 and 128)

[4] A. S. Almgren, J. B. Bell, A. Nonaka, and M. Zingale. Low Mach Number Modeling of Type Ia Supernovae. III. Reactions. *APJ*, 684:449–470, 2008. (Cited on pages 132, 136, and 142)

[5] A. S. Almgren, J. B. Bell, C. A. Rendleman, and M. Zingale. Low Mach Number Modeling of Type Ia Supernovae. I. Hydrodynamics. *APJ*, 637:922–936, February 2006. (Cited on pages 71, 72, 132, 135, and 136)

[6] A. S. Almgren, J. B. Bell, C. A. Rendleman, and M. Zingale. Low Mach Number Modeling of Type Ia Supernovae. II. Energy Evolution. *APJ*, 649:927–938, October 2006. (Cited on pages 132, 135, 136, and 139)

[7] A. S. Almgren, J. B. Bell, and W. G. Szymczak. A numerical method for the incompressible Navier-Stokes equations based on an approximate projection. *SIAM J. Sci. Comput.*, 17(2):358–369, March 1996. (Cited on page 121)

[8] J. B. Bell, P. Colella, and H. M. Glaz. A Second Order Projection Method for the Incompressible Navier-Stokes Equations. *Journal of Computational Physics*, 85:257, December 1989. (Cited on page 120)

[9] J. B. Bell, P. Colella, and L. H. Howell. An efficient second-order projection method for viscous incompressible flow. In *Proceedings of the Tenth AIAA Com-*

putational Fluid Dynamics Conference, pages 360–367. AIAA, June 1991. see also: https://seesar.lbl.gov/anag/publications/colella/A_2_10.pdf. (Cited on page 120)

[10] J. B. Bell, C. N. Dawson, and G. R. Shubin. An unsplit, higher order Godunov method for scalar conservation l aws in multiple dimensions. 74:1–24, 1988. (Cited on pages 43 and 74)

[11] J. B. Bell, M. S. Day, C. A. Rendleman, S. E. Woosley, and M. A. Zingale. Adaptive low Mach number simulations of nuclear flame microphysics. *Journal of Computational Physics*, 195(2):677–694, 2004. (Cited on pages 111, 132, and 135)

[12] J. B. Bell and D. L. Marcus. A second-order projection method for variable-density flows. *Journal of Computational Physics*, 101(2):334 – 348, 1992. (Cited on page 131)

[13] M. J. Berger and P. Colella. Local adaptive mesh refinement for shock hydrodynamics. 82(1):64–84, May 1989. (Cited on page 22)

[14] W. L. Briggs, V-E. Henson, and S. F. McCormick. *A Mutigrid Tutorial, 2nd Ed.* SIAM, 2000. (Cited on pages 87 and 96)

[15] G. L. Bryan, M. L. Norman, J. M. Stone, R. Cen, and J. P. Ostriker. A piecewise parabolic method for cosmological hydrodynamics. *Computer Physics Communications*, 89:149–168, August 1995. (Cited on page 85)

[16] G. D. Byrne and A. C. Hindmarsh. Stiff ODE Solvers: A Review of Current and Coming Attractions. *UCRL-94297 Preprint*, 1986. (Cited on page 11)

[17] A. J. Chorin. Numerical solution of the Navier-Stokes equations. *Math. Comp.*, 22:745–762, 1968. (Cited on page 120)

[18] P. Colella. A direct Eulerian MUSCL scheme for gas dynamics. *SIAM J Sci Stat Comput*, 6(1):104–117, 1985. (Cited on page 33)

[19] P. Colella. Multidimensional upwind methods for hyperbolic conservation laws. *Journal of Computational Physics*, 87:171–200, March 1990. (Cited on pages 33, 42, 43, 63, and 123)

[20] P. Colella and H. M. Glaz. Efficient solution algorithms for the Riemann problem for real gases. *Journal of Computational Physics*, 59:264–289, June 1985. (Cited on pages 62, 69, and 82)

[21] P. Colella and E. G. Puckett. *Modern Numerical Methods for Fluid Flow*. unpublished manuscript. obtained from http://www.amath.unc.edu/Faculty/minion/class/puckett/. (Cited on page 102)

[22] P. Colella and M. D. Sekora. A limiter for PPM that preserves accuracy at smooth extrema. *Journal of Computational Physics*, 227:7069–7076, July 2008. (Cited on page 68)

[23] P. Colella and P. R. Woodward. The Piecewise Parabolic Method (PPM) for Gas-Dynamical Simulations. *Journal of Computational Physics*, 54:174–201, September 1984. (Cited on pages 63, 64, and 80)

[24] J. P. Cox and R. T. Giuli. *Principles of Stellar Structure*, volume 1. Cambridge University Press, 1968. (Cited on pages 72 and 73)

[25] M. S. Day and J. B. Bell. Numerical simulation of laminar reacting flows with complex chemistry. *Combust. Theory Modelling*, 4(4):535–556, 2000. (Cited on pages 132 and 135)

[26] D. R. Durran. Improving the anelastic approximation. 46(11):1453–1461, 1989. (Cited on page 135)

[27] A. Garcia. *Numerical Methods for Physics, 2nd Edition*. Addison-Wesley, 1999. (Cited on pages 2 and 7)

[28] David Goldberg. What every computer scientist should know about floating point arithmetic. *ACM Computing Surveys*, 23(1):5–48, 1991. (Cited on page 2)

[29] C. J. Hansen, S. D. Kawaler, and V. Trimble. *Stellar interiors : physical principles, structure, and evolution*. 2004. (Cited on page 134)

[30] Rupert Klein and Olivier Pauluis. Thermodynamic consistency of a pseudoincompressible approximation for general equations of state. March 2012. (Cited on pages 136 and 139)

[31] C. B. Laney. *Computational Gasdynamics*. Cambridge, 1998. (Cited on page 32)

[32] Randall J. LeVeque. *Finite-Volume Methods for Hyperbolic Problems*. Cambridge University Press, 2002. (Cited on pages 32, 33, 43, 48, and 56)

[33] C. M. Malone, A. Nonaka, A. S. Almgren, J. B. Bell, and M. Zingale. Multidimensional Modeling of Type I X-ray Bursts. I. Two-dimensional Convection Prior to the Outburst of a Pure ^{4}He Accretor. *ApJ*, 728:118, February 2011. (Cited on page 111)

[34] D. F. Martin and P. Colella. A Cell-Centered Adaptive Projection Method for the Incompressible Euler Equations. *Journal of Computational Physics*, 163:271–312, September 2000. (Cited on page 121)

[35] S. May, A. J. Nonaka, A. S. Almgren, and J. B. Bell. An unsplit, higher order Godunov method using quadratic reconstruction for advection in multiple dimensions. *Communications in Applied Mathematics and Computational Science*, 6(1), 2011. (Cited on page 43)

[36] P. McCorquodale and P. Colella. A high-order finite-volume method for conservation laws on locally refined grids. *Communication in Applied Mathematics and Computational Science*, 6(1):1–25, 2011. (Cited on page 79)

[37] G. H. Miller and P. Colella. A Conservative Three-Dimensional Eulerian Method for Coupled Solid-Fluid Shock Capturing. *Journal of Computational Physics*, 183:26–82, November 2002. (Cited on pages 62, 63, and 81)

[38] M. L. Minion. A Projection Method for Locally Refined Grids. *Journal of Computational Physics*, 127:158–177, 1996. (Cited on page 130)

[39] J. J. Monaghan. An introduction to SPH. *Computer Physics Communications*, 48:89–96, January 1988. (Cited on page 15)

[40] A. Nonaka, A. S. Almgren, J. B. Bell, M. J. Lijewski, C. M. Malone, and M. Zingale. `MAESTRO`:an adaptive low mach number hydrodynamics algorithm for stellar flows. *APJS*, 188:358–383, 2010. (Cited on pages 136 and 139)

[41] T. Pang. *An Introduction to Computational Physics, 2nd Edition*. Cambridge, 2006. (Cited on page 2)

[42] R. B. Pember, L. H. Howell, J. B. Bell, P. Colella, W. Y. Crutchfield, W. A. Fiveland, and J. P. Jessee. An adaptive projection method for unsteady low-Mach number combustion. *Comb. Sci. Tech.*, 140:123–168, 1998. (Cited on pages 132 and 135)

[43] T. Plewa and E. Müller. The consistent multi-fluid advection method. *Astron Astrophys*, 342:179–191, February 1999. (Cited on page 81)

[44] W. J. Rider. Approximate projection methods for incompressible flow: Implementation, variants and robustness. Technical report, LANL UNCLASSIFIED REPORT LA-UR-94-2000, LOS ALAMOS NATIONAL LABORATORY, 1995. (Cited on page 128)

[45] J. Saltzman. An Unsplit 3D Upwind Method for Hyperbolic Conservation Laws. *Journal of Computational Physics*, 115:153–168, November 1994. (Cited on pages 63, 79, and 86)

[46] C. W. Schulz-Rinne, J. P. Collins, and H. M. Glaz. Numerical Solution of the Riemann Problem for Two-Dimensional Gas Dynamics. *SIAM J Sci Comput*, 14(6):1394–1414, 1993. (Cited on page 155)

[47] V. Springel. E pur si muove: Galilean-invariant cosmological hydrodynamical simulations on a moving mesh. 401:791–851, January 2010. (Cited on page 22)

[48] J. M. Stone, T. A. Gardiner, P. Teuben, J. F. Hawley, and J. B. Simon. Athena: A New Code for Astrophysical MHD. *Astrophys J Suppl S*, 178:137–177, September 2008. (Cited on pages 54 and 85)

[49] G. Strang. On the construction and comparison of difference schemes. *SIAM Journal on Numerical Analysis*, 5(3):pp. 506–517, 1968. (Cited on pages 40 and 114)

[50] F. X. Timmes. Integration of Nuclear Reaction Networks for Stellar Hydrodynamics. *APJS*, 124:241–263, September 1999. (Cited on page 11)

[51] E. F. Toro. *Riemann Solvers and Numerical Methods for Fluid Dynamics*. Springer, 1997. (Cited on pages 33, 43, 54, 56, 68, and 124)

[52] Geoffrey M. Vasil, Daniel Lecoanet, Benjamin P. Brown, Toby S. Wood, and Ellen G. Zweibel. Energy conservation and gravity waves in sound-proof treatments of stellar interiors. ii. lagrangian constrained analysis. 773:169–, 2013. (Cited on page 139)

[53] Natalia Vladimirova, V. Gregory Weirs, and Lenya Ryzhik. Flame capturing with an advection-reaction-diffusion model. *Combustion Theory and Modelling*, 10(5):727–747, 2006. (Cited on page 115)

[54] S. Yakowitz and F. Szidarovszky. *An Introduction to Numerical Computations, 2nd edition*. Prentice Hall, 1989. (Cited on page 5)

[55] M. Zingale. pyro: A teaching code for computational astrophysical hydrodynamics. *Astronomy and Computing*, 2014. accepted for publication. (Cited on pages xiv and 153)

[56] M. Zingale, L. J. Dursi, J. ZuHone, A. C. Calder, B. Fryxell, T. Plewa, J. W. Truran, A. Caceres, K. Olson, P. M. Ricker, K. Riley, R. Rosner, A. Siegel, F. X. Timmes, and N. Vladimirova. Mapping Initial Hydrostatic Models in Godunov Codes. *Astrophys J Suppl S*, 143:539–565, December 2002. (Cited on page 77)